Refine Search

Search Results -

Terms	Documents
L4 and ("sgml" or "standard generalized markup language")	2

Database:

US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search:

Refine Search

Recall Text Clear Interrupt

Search History

DATE: Monday, October 09, 2006 **Purge Queries** Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
	DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR		
L10	l4 and ("sgml" or "standard generalized markup language")	2	L10
L9	L6 and (highlight$ or color$ or colour$) near (text or characters or font)	1	L9
L8	L7 and (highlight$ or color$ or colour$) near (text or characters or font)	0	L8
L7	L6 and structured near document	2	L7
L6	l3 and ("sgml" or "standard generalized markup language")	6	L6
L5	5933841.uref.	24	L5
	DB=USPT; PLUR=YES; OP=OR		
L4	("5933841")[URPN]	24	L4
L3	(5297249 | 5339091 | 5623679 | 5572643 | 5146552 | 5774109 | 5634064 | 4829453 | 5432903 | 5021989 | 5142678 | 5129082 | 5140521 | 5331547 | 5708826 | 5404506 | 5329111 | 5530852 | 4752908 | 4616336 | 5278980 | 5204947 | 5339433 | 5625781 | 5428776 | 5557722 | 5392387)![PN]	27	L3
L2	("5933841")[PN]	1	L2

DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR

L1	5933841.pn.	2	L1

END OF SEARCH HISTORY

Refine Search

Search Results -

Terms	Documents
707/529	241

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search:

Refine Search

Recall Text Clear Interrupt

Search History

DATE: Monday, October 09, 2006 **Purge Queries** Printable Copy Create Case

Set Name side by side	**Query**	**Hit Count**	**Set Name** result set
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L67	707/529	241	L67
L66	715/529	90	L66
L65	715/513	2638	L65
L64	707/513	2747	L64
DB=USPT; PLUR=YES; OP=OR			
L63	'4703516'.pn.	1	L63
L62	'4703516'.pn.	1	L62
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L61	L56 and (highlight$ or color$ or colour$) near2 (font or text)	30	L61
DB=USPT; PLUR=YES; OP=OR			
L60	'5557678'.pn.	1	L60
L59	'5666215'.pn.	1	L59
L58	'5784461'.pn.	1	L58
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			

L57	L56 and (highlight or color or colour) near2 (font or text)	27	L57
L56	L55 and (character with string$ or term)	211	L56
L55	"structured document".ti.ab.	858	L55
L54	382/180	714	L54
L53	382.clas.	55980	L53
L52	707.clas.	38131	L52
L51	715.clas.	26980	L51
L50	715/513	2638	L50
L49	707/513	2747	L49
L48	707/3	9365	L48
L47	707/2	5690	L47
L46	707/1	8541	L46
DB=USPT; PLUR=YES; OP=OR			
L45	'5261040'.pn.	1	L45
L44	'4807182'.pn.	1	L44
DB=EPAB; PLUR=YES; OP=OR			
L43	EP-747836-A1.did.	1	L43
L42	EP-747836-A1.did.	1	L42
DB=USPT; PLUR=YES; OP=OR			
L41	'5261040'.pn.	1	L41
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L40	5956726.pn.	2	L40
DB=USPT; PLUR=YES; OP=OR			
L39	'5953716'.pn.	1	L39
L38	'5953716'.pn.	1	L38
L37	'5970472'.pn.	1	L37
L36	'5983267'.pn.	1	L36
L35	'5983267'.pn.	1	L35
L34	'6032130'.pn.	1	L34
L33	'6102969'.pn.	1	L33
L32	'6128619'.pn.	1	L32
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L31	L29 not @py>1997	25	L31
L30	L29 not @py>1996	16	L30
L29	L28 and ("document type definition" or "dtd")	967	L29
L28	("sgml" or "standard generalized markup language")	4539	L28
L27	6199082.pn.	2	L27
L26	6199082.uref.	85	L26
DB=USPT; PLUR=YES; OP=OR			
L25	'4599691'.pn.	1	L25
L24	'4949253'.pn.	1	L24

L23	'5269014'.pn.	1	L23
L22	'5530863'.pn.	1	L22
L21	'5321606'.pn.	1	L21
L20	'5940075'.pn.	1	L20
L19	'5587902'.pn.	1	L19
L18	'5655130'.pn.	1	L18
L17	'5669005'.pn.	1	L17
L16	'5669007'.pn.	1	L16
L15	'5694609'.pn.	1	L15
L14	'6014680'.pn.	1	L14
L13	'5920879'.pn.	1	L13
L12	'5920879'.pn.	1	L12
L11	'6014680'.pn.	1	L11
L10	'5956726'.pn.	1	L10
L9	'5956726'.pn.	1	L9
L8	'6014680'.pn.	1	L8
L7	'6014680'.pn.	1	L7
L6	'6073143'.pn.	1	L6
L5	'6073143'.pn.	1	L5
L4	(5920879 | 5911776 | 5915259 | 5752021 | 6014680 | 5802529)![PN]	6	L4
L3	("6202072")[PN]	1	L3
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L2	6202072.uref.	15	L2
L1	6202072.pn.	3	L1

END OF SEARCH HISTORY

Sign in

Web Images Video[New!] News Maps more »

Google standard generalized markup language 1996 " Search Advanced Search Preferences

Web Results **1** - **10** of about **24,700** for **standard generalized markup language 1996 "structured document**

Scholarly articles for **standard generalized markup language 1996 "structured document"**

Index structures for structured documents - Lee - Cited by 79
The HL7 Clinical Document Architecture - Dolin - Cited by 105
XML: a door to automated Web applications - Khare - Cited by 99

Cover Pages: **General** SGML/XML Applications
"HTF is defined in terms of the ISO **Standard Generalized Markup Language** (SGML). SGML is a sort of meta-**standard** for defining **structured document** types and ...
xml.coverpages.org/gen-apps.html - 238k - Cached - Similar pages

Cover Pages: Extensible **Markup Language** (XML)
XML is a simplified subset of the **Standard Generalized Markup Language ... Markup Language**), a **standard** for **structured document** interchange on the Web, ...
xml.coverpages.org/xml.html - 426k - Cached - Similar pages
[More results from xml.coverpages.org]

Title Index
... Expressiveness of **Structured Document** Query Languages Based on Attribute ... Text and Office Systems — **Standard Generalized Markup Language** (SGML) ...
dret.net/biblio/titles - 950k - Cached - Similar pages

Standard Generalized Markup Language: Mathematical and ...
The **Standard Generalized Markup Language** SGML an ISO **standard**, ... Extensions of Attribute Grammars for **Structured Document** Queries - Neven (1999) (Correct) ...
citeseer.ist.psu.edu/wood95**standard**.html - 26k - Cached - Similar pages

Citations: Information Processing --- Text and Office Systems ...
Tables present a challenging problem for **structured document** modelers as they do ...
Chapter 2 SGML HyTime Overview The **Standard Generalized Markup language** ...
citeseer.ist.psu.edu/context/21103/0 - 35k - Cached - Similar pages
[More results from citeseer.ist.psu.edu]

DSSSL - Document Style Semantic and Specification **Language**
DSSSL describes how such a **structured document** might be presented ... The original **Standard Generalized Markup Language** (SGML) style sheet **language**. ...
www.auditmypc.com/acronym/DSSSL.asp - 36k - Cached - Similar pages

Computers & Texts 15: Burnard
Readers may not need to be reminded what SGML is (**Standard Generalized Markup Language**: the international **standard** for **structured document** interchange, ...
users.ox.ac.uk/~ctitext2/publish/comtxt/ct15/burnard.html - 21k - Cached - Similar pages

An Overview of the Extensible **Markup Language** - Jun 98
HTML is an application of the **Standard Generalized Markup Language** (SGML), defined in ISO ... XML, and **Structured Document** Interchange Activity of the W3C. ...
www.stsc.hill.af.mil/crosstalk/1998/06/xml.asp - 24k - Cached - Similar pages

Extensible **Markup** Lannguage (XML) for Music Applications: An ...

It is a subset of **Standard Generalized Markup Language** (SGML), which was developed in ... Although the formal description was adopted by the ISO in **1996**, ...
www.recordare.com/good/cm12intro.html - 21k - Cached - Similar pages

W3C Activity: SGML, XML, and **Structured Document** Interchange
HTML is based on SGML (**Standard Generalized Markup Language**), ... Presentation of the XML draft at the SGML 96 Conference in Boston, November **1996**. ...
www.w3.org/XML/Activity-19970610 - 12k - Cached - Similar pages

Gooooooooooogle ▶
Result Page: 1 2 3 4 5 6 7 8 9 10 **Next**

Free! Speed up the web. Download the Google Web Accelerator.

standard generalized markup langua Search

Search within results | Language Tools | Search Tips | Dissatisfied? Help us improve

Google Home - Advertising Programs - Business Solutions - About Google

Sign in

Web Images Video New! News Maps more »

standard generalized markup language 1996 " Search Advanced Search Preferences

Web Results **31** - **40** of about **24,700** for **standard generalized markup language 1996 "structured documer**

POSC - XML for E&P - Glossary
GCA developed and fosters the **Standard Generalized Markup Language** (SGML), from which the ... A data format for **structured document** interchange on the Web. ...
www.posc.org/ebiz/resources/glossaryMain.html - 26k - Cached - Similar pages

Beyond Web Technology - Lessons Learnt from BSCW
HTML [3] is a SGML application conforming to International **Standard** ISO 8879 - **Standard Generalized Markup Language** [8]. It acts as the universal document ...
bscw.fit.fraunhofer.de/Papers/wetice98/index.html - 28k - Cached - Similar pages

The Role of XML in Open Hypermedia Systems
Table 1: **Structured document standards** on the Web ... Currently, HTML is used as the common **markup** and linking **language** on the Web. ...
www.aue.auc.dk/~kock/OHS-HT98/Papers/ossenbruggen.html - 25k -
Cached - Similar pages

Hyper-activity in the Web-world
... (**Standard Generalised Markup Language**) **standard** for **structured document** interchange ... At the end of **1996** the W3C SGML workgroup started to develop XML ...
cern.ch/cnlart/227/art_xml.html - 16k - Cached - Similar pages

Keystone Websites: Glossary
SGML: The **Standard Generalized Markup Language** (SGML) is a metalanguage in which one can define **markup** languages for documents. ...
keystonewebsites.com/glossary/ - 39k - Cached - Similar pages

Extensible **Markup Language** (XML) - Summer Institute for ...
SGML, XML, and **Structured Document** Interchange - W3C activity statement ... The **Standard Generalized Markup Language** and the HyperText **Markup Language** are ...
www.censa.org/html/weblinks/**Standards**-Relevant-to-CENSA/xml-sil-pages.htm - 301k -
Cached - Similar pages

SurfWax: News, Reviews and Articles On SGML
It reaches back into XML's background in the **Standard Generalized Markup Language**, or SGML, and also covers related parsing tools, programming languages, ...
news.surfwax.com/webservices/files/SGML.html - 26k - Cached - Similar pages

XML Primer
This became SGML (**Standard Generalized Markup Language**) and was moved to ISO to ... This is W3C activity statement for SGML, XML, and **Structured Document** ...
www.w3c.rl.ac.uk/primers/xml/xmlprimer.htm - 112k - Cached - Similar pages

[PDF] **Structured Document** Framework for Design Patterns Based on SGML
File Format: PDF/Adobe Acrobat
Wesley (**1996**). [8] F.Buschmann and R. Meunier, "A System of Patterns",. in [6]. [9] ISO 8879 **Standard Generalized Markup Language**. (SGML) (1986). ...
doi.ieeecomputersociety.org/10.1109/CMPSAC.1997.624848 - Similar pages

Structured document browser - Patent 5933841

One of the first **markup** tools was the **Standard Generalized Markup Language** (SGML). SGML was developed by the International **Standards** Organization and has ...
www.freepatentsonline.com/5933841.html - 73k - Cached - Similar pages

◀ Gooooooooooooogle ▶

Result Page: **Previous** 1 2 3 4 5 6 7 8 9 10 11 12 13 **Next**

Free! Speed up the web. Download the Google Web Accelerator.

---

standard generalized markup langua Search

Search within results | Language Tools | Search Tips

Google Home - Advertising Programs - Business Solutions - About Google

Site Contents

Search Patents
Use our search engine to find what you need

Data and Analytical Services
Complete custom solutions

Syntax Reference
Learn our powerful search syntax

F.A.Q.
About this site and our patent search engine

Crazy Patents
People patented these???

RSS Feeds
Subscribe to our RSS Feeds

Title:	**Structured document browser**
Document Type and Number:	United States Patent 5933841
Link to this Page:	http://www.freepatentsonline.com/5933841.html
Abstract:	A structured document browser includes a constant user interface for that is organized according to a pre-defined structure. The structured been marked with embedded codes that specify the structure of the c set of icons. When the icon is selected while browsing a document, th corresponding to the icon selected, while preserving the constant use

Inventors:	Schumacher, Robert M.; Matthews, James E.;
Application Number:	649271
Filing Date:	1996-05-17
Publication Date:	1999-08-03
View Patent Images:	**View PDF Images**
Related Patents:	**View patents that cite this patent**
Export Citation:	**Click for automatic bibliography generation**
Assignee:	Ameritech Corporation (Hoffman Estates, IL)
Current Classes:	**715 / 501.1**
International Classes:	G06F 017/21
Field of Search:	395/761,762,773,774,776,777,335,347,348-349,354 345/172,335,3 707/500,501,512,513,514,515,517

US Patent References:

4616336	Oct., 1986	Robertson et al.	**395 / 773**.
4752908	Jun., 1988	Bouillot.	
4829453	May., 1989	Katsuta et al.	**395 / 773**.
5021989	Jun., 1991	Fujisawa et al.	**345 / 350**.
5129082	Jul., 1992	Tirfing et al.	
5140521	Aug., 1992	Kozol et al.	
5142678	Aug., 1992	MacPhail	**395 / 761**.
5146552	Sep., 1992	Cassorla et al.	**395 / 773**.
5204947	Apr., 1993	Bernstein et al.	
5278980	Jan., 1994	Pedersen et al.	
5297249	Mar., 1994	Bernstein et al.	
5329111	Jul., 1994	Sonoda et al.	
5331547	Jul., 1994	Laszlo.	
5339091	Aug., 1994	Yamazaki et al.	
5339433	Aug., 1994	Frid-Nielsen.	
5392387	Feb., 1995	Fitzpatrick et al.	
5404506	Apr., 1995	Fujisawa et al.	
5428776	Jun., 1995	Rothfield.	
5432903	Jul., 1995	Frid-Nielsen.	
5530852	Jun., 1996	Meske, Jr. et al.	**395 / 600**.
5557722	Sep., 1996	DeRose et al.	**395 / 762**.
5572643	Nov., 1996	Judson	**395 / 793**.
5623679	Apr., 1997	Rivette et al.	**707 / 526**.

5625781	Apr., 1997	Cline et al.	**395 / 335**.
5634064	May., 1997	Warnock et al.	**395 / 774**.
5708826	Jan., 1998	Ikeda et al.	**707 / 513**.
5774109	Jun., 1998	Winsky et al.	**707 / 501**.

Other References: Classified Search and Image Retrieval Student Manual for the Autom May, 1991.
Fowler et al, "Visualizing and Browsing WWW Semantic Content", Em Communication, 1996 Conference, pp. 110-113, 1996.
Harger, "Introducing DSP with an Electronic Book in a Computer Clas pp. 173-179, May 1996.
Gershon, "Moving Happily through the World Wide Web", IEEE Comp 72-75, Mar. 1996.
Simpson, "Mastering WordPerfect 5.1 &5.2 for Windows", pp. 510-53

Primary Examiner: Feild; Joseph H.

Attorney, Agent or Firm: Brinks Hofer Gilson & Lione

Claims:

We claim:

1. In a computer, a browser for viewing documents having embedded codes that ider predefined document structure, said browser comprising:

a user interface comprising a display window that displays a document to a user;

a plurality of input devices;

a first plurality of display regions that are responsive to said input devices, said displa displayed as part of the user interface automatically and configured to correspond to structure regardless of what part of the document is in the display window; and

a controller operative to cause a selected part of the document to be displayed in the input devices to enable one of said display regions that corresponds to the selected p

2. A browser as claimed in claim 1 wherein said plurality of input devices comprises:

a screen pointer that moves on the display window of the user interface and maintair controlled pointing device; and

a selecting device that selects the current position of the screen pointer when enable

wherein the controller causes a selected part of the document to be displayed on the pointer to the display region that corresponds to the selected part and enables the se

3. A browser as claimed in claim 2 wherein said plurality of input devices further com

a first set of keys on a keyboard, each of said keys configured to highlight one of saic display regions as one of said first set of keys is pressed; and

at least one select key on the keyboard configured to select the highlighted region wh

wherein the controller causes a selected part of the document to be displayed on the one select key while the display region that corresponds to the selected part is highlic

4. A browser as claimed in claim 2 further comprising a document menu that lists doc moving the screen pointer to the document menu and enabling the selecting device.

5. A browser as claimed in claim 4 further comprising:

a next part region that has been predefined to display a next part in the document wl moving the screen pointer to the next part region and enabling the selecting device;

a previous part region that has been predefined to display a previous part in the docu screen pointer to the previous part region and enabling the selecting device.

6. A browser as claimed in claim 4 further comprising a region on the user interface t opening a new display window in which the user can input a note.

7. A browser as claimed in claim 1 wherein said plurality of input devices comprises:

a set of keys on a keyboard, each of said keys configured to correspond to a specific

wherein the controller causes a specific part of the document to be displayed in the d corresponds to the specific part.

8. A browser as claimed in claim 1 wherein the embedded codes that identify parts of definition that has been prepared according to the Standard Generalized Markup Lanc

9. A browser as claimed in claim 1 wherein the embedded codes that identify parts of conform to the Hyper-Text Markup Language (HTML).

10. A browser as claimed in claim 1 wherein the embedded codes are elements of SG according to an SGML document type definition.

11. A browser as claimed in claim 10 further comprising a document type menu that predefined structure that can be selected to alter the document menu to list documer document type selected by moving the screen pointer to the document type menu an

12. A browser as claimed in claim 1, further comprising a second plurality of display r said display regions of said second plurality configured to correspond to respective pa response to the part of the document that is displayed in the display window.

13. The browser as claimed in claim 1, wherein the first plurality of display regions ar interface without prompting by the user.

14. The browser as claimed in claim 1, wherein the first plurality of display regions re after a user enables one of said display regions.

15. In a computer, a method for browsing a document within the context of a predefi

initializing a browsing tool having document navigation tools that include a first plura displayed as part of the browsing tool during browsing, said display regions of said fir to respective sections of the predefined document structure regardless of what part o display the respective sections;

displaying a document; and

enabling one of said plurality of display regions to display the respective section.

16. The method of browsing a document as claimed in claim 15 wherein the plurality on a keyboard.

17. The method of browsing a document as claimed in claim 15, further comprising tl

initializing a browsing tool having document navigation tools that include a second plı said second plurality corresponding to respective sections of the predefined documen that is displayed, and operative to display the respective section.

18. The method of browsing a document as claimed in claim 15, wherein the first plu displayed as part of the browsing tool without prompting by the user.

19. The method of browsing a document as claimed in claim 15, wherein the first plu the browsing tool even after a user enables one of said display regions.

20. In a computer comprising a graphical user interface, a method for browsing a doc document structure comprising the steps of:

initializing a browsing tool having document navigation tools that include a first plura first plurality being continuously displayed as part of the browsing tool during browsir respective parts of the predefined document structure regardless of what part of the

displaying a document in a display window; and

viewing parts of the document by repeating the steps of:

moving a screen pointer that maintains a current position on the display window resp selected display region of the document navigation tools configured to correspond to structure; and

selecting the selected display region by enabling a selecting device.

21. The method of browsing a document as claimed in claim 20, further comprising tl

initializing a browsing tool having document navigation tools that include a second plı said second plurality configured to correspond to respective parts of the predefined d document that is displayed in the display window.

22. The method of browsing a document as claimed in claim 20, wherein the first plu displayed as part of the browsing tool without prompting by the user.

23. The method of browsing a document as claimed in claim 20, wherein the first plu the browsing tool even after selection of a display region.

24. In a computer with graphical user interface capabilities, a method for browsing a document structure comprising the steps of:

creating one or more documents having the predefined document structure;

embedding codes in the documents to identify parts of the predefined document strui

initializing a browsing tool having document navigation tools that include a first plura first plurality being continuously displayed as part of the browsing tool during browsir respective parts of the predefined document structure regardless of what part of the

displaying a document in a display window; and

viewing parts of the document by repeating the steps of:

moving a screen pointer that maintains a current position on the display window resp selected region of the document navigation tools configured to correspond to a corres structure;

maintaining said selected display region accessible regardless of what part of the doc

selecting the selected region by enabling a selecting device.

25. A method for browsing a document as claimed in claims 15, 20 or 24 wherein the the steps of:

displaying a next part in the predefined document structure of the document by repe

moving the screen pointer over a next part region of the document navigation tools c follows a part of the document that is currently displayed in the display window; and

selecting the next part region by enabling the selecting device; and

displaying a previous part in the predefined document structure of the document by r

moving the screen pointer over a previous part region of the document navigation to that follows a part of the document that is currently displayed in the display window;

selecting the previous section region by enabling the selecting device.

26. The method of browsing a document as claimed in claim 15, further comprising tl

initializing a browsing tool having document navigation tools that include a second pl said second plurality configured to correspond to respective parts of the predefined d document that is displayed in the display window.

27. The method of browsing a document as claimed in claim 24, wherein the first plu displayed as part of the browsing tool without prompting by the user.

28. The method of browsing a document as claimed in claim 24, wherein the first plu the browsing tool even after selection of a display region.

29. In a computer comprising a graphical user interface, a browser having a user inte codes that identify parts of documents according to a predefined document structure,

a display window that displays a portion of the document to a user;

a screen pointer that moves on the user interface and maintains a current position re

a selecting device that selects the current position of the screen pointer when enable

a document menu that lists documents that the user can select for viewing by movin enabling the selecting device;

a document type menu that lists at least one document type having a predefined stru menu to list documents that conform to the predefined structure of the document typ document type menu and enabling the selecting device;

a first plurality of display regions on the user interface, said display regions of said fir to correspond to a respective part of the predefined document structure regardless o window; and

a controller operative to cause a selected part of the document to be displayed in the pointer to the region that corresponds to the selected part and enables the selecting

30. A browser as claimed in claim 29, further comprising a second plurality of display of said second plurality configured to correspond to a respective part of the predefine the document that is displayed in the display window.

Description:

A portion of the disclosure of this document contains material which is subject to cop

objection to the facsimile reproduction by any one of the patent disclosure as it appe
or records, but otherwise reserves all rights to copyright protection whatsoever.

FIELD OF THE INVENTION

This invention relates to computer applications for viewing documents, and in particu
having a predefined structure.

BACKGROUND OF THE INVENTION

Current computing environments typically include a graphical user interface (GUI). IE
either OS/2 or Windows. The Macintosh has always had GUI capabilities as part of its
including those that run Unix or VAX/VMS operating systems, are available with com

GUI's are generated using a set of software tools that put graphical objects on the co
screen pointer that the user controls with a mouse or a trackball. The user moves the
selected objects on the screen. The user can select an object by using a selecting de
an object, the user instructs the operating system or an application to execute the fu
can include the graphical representation of buttons, menus or any other graphic obje

GUI's are the foundation of hypertext and hypermedia applications. Such applications
which graphical objects are configured to correspond, or to link to objects of informat
having the graphical representation of a button to display a motion video by selecting
button can be configured to make a sound, display an image or display a separate te

Today's increasing interest in the Internet is due in part to the improvements in hype
growing dramatically due to the evolution of standard markup languages that allow u
documents, and of presenters or viewers that interpret the markup language in the d

One of the first markup tools was the Standard Generalized Markup Language (SGML
Standards Organization and has been adopted by the Department of Defense and oth
documentation. SGML is machine-based in a manner similar to a computer language.
be defined according to the specifications of SGML for a given document structure. Th
document. The documents are then viewed using a viewer or browser that interprets
data structure defined in the DTD.

The HyperText Markup Language (HTML) is an instance of a DTD defined by SGML. H
documents for use with HTML browsers. Browsers display the documents according t
The stylesheets contain rules or instructions that dictate the appearance of a docume
also contain references to other documents in different computers. These references
regions of the documents such that a user retrieves the document reference by selec
one document to others in a meaningful way such that a viewer may provide a user v
linked together in a web-like fashion.

One common characteristic of many browsers is that the links to information are pres
the user to other documents or to locations within the same document, but typically,
within the documents.

Having the control to the information links within the documents themselves is adequ
purpose is to obtain information in brief, concise statements. But where a document i
document since the only potential access to other destinations are in whatever part o

Moreover, organizations often work with standardized documents. These documents t
usually characterized by a standard structure. These documents may be long and the
to access information found in a specific section of the well-known structure of the dc

SUMMARY OF THE INVENTION

In view of the above, a structured document browser is provided with a user interfac
browses documents according to their structure instead of their contents. The browse
identify sections of the structure of the document. In one preferred embodiment, the

configured to correspond to respective parts of the predefined document structure re displayed. In another preferred embodiment, the browser further includes a second p to respective parts of the predefined document structure in response to the part of th

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a preferred embodiment of the structured document bro components of the computer.

FIG. 2 illustrates a sample structure for a document.

FIG. 3 illustrates a portion of a document having the structure shown in FIG. 2 after i

FIG. 4 illustrates an SGML document type definition (DTD) created for use by a struc the structure shown in FIG. 2.

FIG. 5 is a representation of an example of a user interface of the structured docume

FIG. 6 is a flow chart showing the process of retrieving a structured document and ill response to the selection of a button.

FIG. 7 illustrates an example of a bks data structure.

FIG. 8 is a diagram that shows the interaction between a button, a map file and bit m

FIGS. 9A & 9B illustrate the operation of the browser of FIG. 1.

FIG. 10 illustrates one example of an alternative implementation of the user interface

FIG. 11 illustrates a second alternative implementation of the user interface.

FIG. 12 illustrates a third alternative implementation of the user interface.

FIG. 13 illustrates a fourth alternative implementation of the user interface.

DETAILED DESCRIPTION OF THE PRESENTLY PREFERRED EMBODIMENTS

In the description that follows, reference is made to the drawings where like element

A presently preferred embodiment of this invention includes an application program c the `c` programming language in the Windows environment. The browser.exe execu three dynamic link libraries named sit.dll, cgrmzv.dll and ct13d.dll. These libraries ar interface development system from Electronic Book Technologies, Inc. These Dynate SGML related functions and the graphic input/output functions. Further information r by contacting Electronic Book Technologies, Inc. at One Richmond Square, Providenc

The browser.exe program uses data structures in several support files that are in the support files will be described in more detail in the description that follows.

A listing of the present version of browser.exe is attached as Appendix I of this specif browser.exe. The presently preferred embodiment may be carried out by converting t that are well known in the art. After conversion, the browser.exe file may be execute libraries and the support files described in this disclosure.

It is to be understood however, that an embodiment of the present invention may be any suitable development system.

The browser application in a presently preferred embodiment is referred to in the foll shown in FIG. 1. FIG. 1 is a block diagram describing at a high level the browser 80 i the browser 80 in its operating environment include the browser 80 itself, an operati 108, a keyboard 110, a pointing device 100, a selecting device 102 and a monitor 11

I/O system 114 and a GUI system 116.

The double headed arrows 118 denote the communication between the respective coı communication over a network where appropriate.

The hardware devices 108, 110, 100, 102, 112 may be implemented by choosing froı pointing device 100 may be implemented using a mouse, a trackball or any other de\ A selecting device 102 is typically implemented with mouse buttons or buttons that o any device that may be used to affect the selection of an object at the location of the 102. A selecting device 102 may even include a key on the keyboard 110. The storag access memory (RAM), the temporary storage out of which programs are executed a: programs are stored. The hardware devices 108, 110, 100, 102, 112 are understood operation in the computing environment.

The browser 80 includes at least a controller 82 and a system interface 84. The syste communication between the browser 80 and the operating system 104, the I/O syste receives and interprets requests from the system interface 84 to perform a browser f receives signals from the I/O system 114 that the pointing device 100 and the selecti icon, to request a display of a section of a document. The controller 82 receives the iı determine which document section to display.

In a presently preferred embodiment, the system interface 84 includes the functions and any operating system or I/O system functions. The controller 82 in a preferred eı browser.exe. It is to be understood that the diagram in FIG. 1 is by way of illustratioı structure chosen to carry out the invention.

The browser 80 operates with documents that have been prepared as described belo\ documents according to their structure, the utility of the browser 80 is maximized wh structure for its key documents. A software engineering group, for example, may finc the software requirement specifications that the group develops. A different structure and yet a different structure works for the group's test documentation. The group's g uniformity.

Referring to FIG. 2, a marketing group might maintain product descriptions for its coı predefined document structure 10. This structure is predefined to have headings 12 t information, product availability information, ordering information, billing information information. The structure also has sub-headings 14 within each heading where relev for sections devoted to a product description, aliases, product features and instructioı

A specific document of the predefined structure in FIG. 2 is marked with codes for vie embodiment, codes are used to mark the document as shown in FIG. 3. The codes sh In a preferred embodiment, the codes are SGML elements. These codes may be repla alternative embodiments. In the marked document 20, the application identification c marked for use by the browser 80. The overall document structure is identified with a component is then marked with an appropriate code or element.

The sections in a document are preferably marked according to a convention. First, tl element. For example, the <OVER> element 26 identifies the beginning of the sectioı product. The <OVER> element 26 is followed by the heading and text that constitute section name element 28 indicates that the information that follows the element 28 s of the same structure for that section. In the marked document 20 in FIG. 3, the nan Immediately after the name, an end section name element 32 indicates the end of th begin section name element 28 and the end section name element 32 is the text 30 t section element 34 indicates the end of the section. In this case, the element </OVEI convention of marking the beginning and the end of parts of the document is used to 42) that form the standard document structure.

Begin sub-section elements 38 and end sub-section elements 42 mark the sub-sectio with the stipulation that the end elements 39 are in an order that keeps the sub-secti have sub-sections within sub-sections.

In a preferred embodiment, the structure of documents is defined by a document typ how a document structure is represented in SGML. Codes such as those referenced a section or structure part of the document as shown in FIG. 3, are defined by setting a 4; setting a heading name, as shown at 47 and 49; and listing sub-parts as shown at 24 in FIG. 3 is defined in the DTD 40 as the element name 42 for the high level docu The list of sections 46 comprises the codes defining the sections of the document. Th 25 are defined in the DTD 40 of FIG. 4 as element names 44 for the respective sectio codes, such as the begin section name element 27 in FIG. 3, is shown as part of a se 45 is also defined at 51. If a section has sub-parts, or sub-sections, the codes for the definition of the section code as shown at 48. Each sub-section code is then defined i syntax and constructs of SGML may be obtained by referring to the ISO Standard for

In a preferred embodiment, a document having a basic word processing format may known as an SGML instance, using an SGML utility. For example, a utility called DYN documents having the structure described in FIG. 2 to create a DTD 40, illustrated in in FIG. 3. Other SGML utilities may be used to create DTD's and SGML instances. The how alternative embodiments might implement a program that interprets SGML DTD'

An infinite number of DTD's may be used, including common DTD's such as the HTML invention may not use SGML as a markup language. Any other suitable markup lang may be used as well, provided that the appropriate software components are availabl invention may be designed to support the use of more than one markup language.

Once a document has been marked and converted into a format that is appropriate f on a computer. Referring back to FIG. 1, the user invokes the browser 80 by using th to select the browser 80 in a manner dictated by the GUI of the operating system 10 system 104 is the Windows Operating System (Version 3.1 and later for purposes of and the selecting device 102 includes a pair of mouse buttons (left and right buttons) on the monitor 112, includes a menu bar 68, icons 66 representing application progr controlled by the user with the mouse pointing device 100.

To initialize the browser 80 in the Windows environment, the mouse 100 may be use icon 64 in the operating system interface 106 and the left mouse button may be dou

Once the browser 80 has been initialized, the graphical user interface changes from t the browser user interface as shown at 50 in FIG. 5.

Referring to FIG. 5, the user interface 50 of the browser 80 gives the user the capabi selecting different sections of the document for display. The objects used to browse t interface 50 regardless of where in a document a user is browsing. The user interface the document structure.

The user interface 50 of the browser 80 includes a document menu 52, a document t row of selectors 58 and a display window 60.

The user selects a document for browsing by using the document menu 52. The docu graphical user interface menu objects such that the user selects a document menu a available. The user then selects the document desired from the menu using the selec pointer 62.

The user interface 50 of the browser 80 is configured in a manner that allows the doc types, to be listed in a document type menu 54. The document type is selected by th document. The name of the document type is the name or alias of directories design consistent structure. When a different document type is selected, the names listed in names of the available documents having the new document type.

The selectors 56, 58 are examples of document navigation tools that may be used fo embodiment. More specifically, selectors 56, 58 are display regions in the user interf operations when the user places the screen pointer 62 over one of the selectors 56, 102. In the presently preferred embodiment, the display regions are depicted as icon images on them. The image may be designed to convey a sense of the operation to b

The first row of selectors 56 is configured to correspond to the first level of sections i
the first row 56 is configured to correspond to the Overview section 16 in the docume

Each selector is configured so that when the user places the screen pointer 62 over tl device 102, the system interface 84 (described above with reference to FIG. 1) recei The system interface 84 may then determine the document section associated with tl section to the controller 82 (described above with reference to FIG. 1). Alternatively, controller 82 the identification of the chosen selector and let the controller 82 determ cause the document to be searched for the document section that matches the select browser 80 displays in the display window 60 the section of the document structure t

Once the desired section is in the display window 60, the user may navigate within th selectors 58. Each time a selector from the first row of selectors 56 is selected, the s correspond to the sub-sections 14 within the section 12 being displayed (as shown in sections within its sub-sections may also be accommodated so that three rows of sel number of sub-sections within sections of a document may be further accommodated system constraints, such as the size of the display window 60.

The user may also navigate within the section by controlling the display window scrol selecting device 102.

To describe the manner in which the selectors 56, 58 in the browser user interface 5(will be described in conjunction with exemplary files or data structures that are utiliz understood that this is only one implementation of the preferred embodiment, and th revised to form different data structures without departing from the scope of the inve tools may be replaced by other functionally equivalent tools.

In the presently preferred embodiment, documents are converted into "books" which directory called the ".backslash.xyz.backslash.books" directory. The terms "book," "c according to the specifications of the Dynatext development system.

In order to create the Dynatext books, the documents that have been coded as illust called DYNATAG which is a component of the Dynatext system. DYNATAG creates a D shown in FIG. 3) of the document. The SGML instance and the DTD are used as input MKBOOK utility creates a binary instance of the document, a directory tree, or a "boo browser 80. For example, the document for Widget having the structure for products MKBOOK to create the book ".backslash.xyz.backslash.books.backslash.widget," a su

The browser 80 uses a number of data files to define how a document is found and di names in the preferred embodiment are 1) the browser executable (browser.exe); 2) dynamic link libraries (sit.dll, cgmzv.dll, & ct13d.dll); 4) a bks file which is an ASCII f library or collection of documents, and names in the browser menus (named *.bks wl 5) a bitmap containing up to 100 regions for icons (named *.bmp or default.bmp) an document element names to the icons in the bmp file and for pop-up text in the exec

The execution of the browser 80 will be described with reference to FIG. 6 which is a document. When the user starts the structured document browser in the manner des system launches the browser.exe executable file as shown at block 120. This file is lo directory. When browser.exe is launched it first looks for the three required dynamic shown at block 122. The DLL's, supplied by Electronic Book Technologies, contain fun processing and access to the books.

If the DLL's are available, the browser then checks for the browser.ini file as shown ir browser 80 reads its contents, as shown at block 126.

As the sample file in Table 1 shows, the browser.ini file contains objects, or data stru □DTEXT! object, and the □MAP! object. The □FILES! object defines an annotation fil from users of the browser regarding the documents being reviewed. The □DTEXT! ot use to find the location of the data directory, security key, and public and private dire of the initial map file that is to be loaded (typically, the name is "default.map"). The r

associations between the document elements and the icons.

TABLE 1

```
Sample browser.ini File

        □Files!
        AnnotationFile=.backslash.xyz.backslash.annot.txt
        □DTEXT!
        DATA.sub.-- DIR=.backslash.xyz.backslash.data
        DTEXT.sub.-- AUTH=@.backslash.xyz.backslash.data.backslash.se
        6
        PUBLIC.sub.-- DIR=.backslash.xyz.backslash.tmp.backslash.publ
        PRIVATE.sub.-- DIR=.backslash.xyz.backslash.tmp.backslash.pri
        □MAP!
        Icons=default.bmp
```

Referring back to FIG. 6, once the browser.ini file is processed at 126, the browser ch default.bks file is an ASCII file which provides the browser with the information requi a document that displays a message of the day.

If a default.bks file is not found, the browser initializes without a book as shown at 1: loaded as shown at 132 and the user interface is presented on the display to the user

The bks files will be described by reference to FIG. 7. The bks files are ASCII files tha be loaded into the menus in the browser and which map file to use. The bks file alwa' The next line identifies the "collection." In a preferred embodiment, a collection, also the books to which the browser 80 has access. In FIG. 7, the collection is located in t

The next line in a bks file as shown in FIG. 7 identifies the collection title 144 which d menu 54. The collection title 144 is an alias for the group of books that will be listed called "Products."

The name of the book 146 is on the next line of the bks file. The book name is actual present under the collection path specified in line two 142 of the bks file. The book til title contains the name the user will see on the document menu 52 (at FIG. 5). The n map file that associates the tag names with the icons in the .bmp file. If specified, th map file. Otherwise, the browser will assume a file name based on a pre-defined nam

Referring back to FIG. 6, a user requests a document 160 by using the document me interface 50 (shown in FIG. 5). When the menu is selected (before a document is sele document and document type names according to the contents of the bks file. When at 160, the browser retrieves the document itself as shown at 162 and the map file a

The browser then verifies that the elements in the map file match the SGML DTD and 166. If there are no discrepancies, the browser 80 reads the bmp file as shown at 16: 168, the browser 80 locates fly-by text for description of icons. The bmp file allows th the selectors 56, 58 (in FIG. 5). The browser also displays the text in the chosen docu

Once the selectors on the user interface match the structure requirements of the doc document to view by pressing a selector button that corresponds to that section as sl selectors 56, 58 and the document structure is established in the map file and in the with reference to FIG. 8.

FIG. 8 shows a portion of the map file 200 for the document having the structure in F interface in which the selectors are placed 220. The first object in the map file is the structure. The next line in the map file is the icon line 182 which is a filename that is the structure corresponding to the map file.

The next line in the map file shown in FIG. 8 contains the □SECTIONS! object 184. T of a set of definitions of data structures 186 that help tie the selector icons to the sec structures indicates the order in which the sections appear in the document.

The first item under □sections! is "RBW-DOC, PROD.NAME, OVER=1: Overview" 188 that the section identified in this line is in the first or highest level section. The OVER used to identify the overview section in the marked structure document shown in FIG icon in the bmp file corresponds to the section identified. The text following the colon messages. For example, when the screen pointer 62 is positioned over an icon 222 a: displayed to indicate the function of the button. The expression OVER=1 192 and the during block 168 in FIG. 6.

An example of the bmp file is shown in FIG. 8 at 210. The icons are stored in the bm| 16 pixels wide by 15 pixels high. In the presently preferred embodiment, the icons m index starts at 1 and proceeds from left to right. So the icon index might appear as f(

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	<etc>							

An example of how the map file relates to the bmp file is given in FIG. 8. Assume the troubleshooting icon, which is the sixth selector or 222 in the top row of selectors 56 corresponds to the sixth icon 224 in the bmp file. The browser 80 refers to the map fi Troubleshooting 226 to determine which element name must be searched. The line a that must be searched when the sixth button in the first row 222 is selected.

While every book could have a separate map file and bmp file, these files relate to all product library should have the same document structure (i.e., document type definit all books in the products (i.e., prd) library and, a prd.bmp provides the icons for all b should relate to the hierarchical structure of the document structure as specified in tr

FIGS. 9A & 9B demonstrate the operation of the browser 80 by illustrating the way in to the pressing of a button. FIG. 9A shows the user interface 50 in an initial state wit document from FIG. 2 in the display window 60. The selectors 58 of the second row a sub-sections 246 of the description section.

The user of the browser 80 may wish to view information that is known to exist in the description. As shown in FIG. 2, the troubleshooting section 18 is towards the end of short as one printed page or long enough to fill several binders. The browser 80 simp access to a pre-defined section of a document by pushing a button. The screen pointe the position over the icon for the troubleshooting button 222. The user then selects tl selecting device 102. The browser 80, using the process illustrated in FIG. 8, then se document for the troubleshooting section. The browser 80 displays the found section In addition to displaying the found section 242, the browser 80 updates the second r sections 248 in the found section 242.

The user interface 50 of the browser 80 may be enhanced by adding objects to give t documents. The user interface 50, shown in FIG. 9A, includes a next section button 9 button 72, a go backward button 74 and string search tools 78, 86, 88, 89. These ob along with the software components that provide the indicated functions.

When the next section button 92 is selected with the combined action of the screen p views the next section in the document. For example, the next section after the Over (See FIG. 2). Selecting the next section button 92 in FIG. 9A causes the browser 80 t button 94 operates in the same manner as the next section button 92 except that the

The go forward button 72 and go backward button 74 may be used to scroll text in tr

The user interface 50 as shown in FIG. 9A may also include string search tools 78, 8€
to input a text string that the user wishes to locate in the document. The next found .
display the locations in the document in which the string was found. The clear search
78.

The feedback entry function gives the user the ability to provide feedback on a docun
later time. As shown in FIG. 9B, by selecting a feedback file in a menu, or by selectin
allow the user to enter a note. The text box 260 may be a compilation of messages t
text box 260 may be saved into a separate repository of data periodically. In a prefer
may be saved to a SGML-based file for support as a document that may be viewed b
the textbox 260 may be replaced by a view of the messages in the display window 6(

It is to be understood that the appearance of the user interface 50 shown in FIGS. 9
present invention. The appearance and the choice of graphic objects may be varied t

Referring to FIG. 10, one example of how the user interface 50 may be altered replac
buttons are merely display regions of the user interface configured to perform a funct
the selecting device 102. In the presently preferred embodiment of FIG. 9A, the sele
FIG. 10, these icons may be replaced with words or phrases 268 that are descriptive

Another variation, shown in FIG. 11, uses a distributed user interface in which the bu
detached from the display window 60. FIG. 11 illustrates the separate windows 50, 2

In another variation shown in FIG. 12, keys on the keyboard 110 may be configured .
to select document section selectors 56, 58. FIG. 12 illustrates a monitor screen 274
In one approach to using the keyboard, the selectors 56, 58 may be mapped to funct
approach which may be combined with the first approach, the browser 80 may first h
as a TAB key 266, or an arrow key 264, and then select the highlighted selector 280
272.

The user interface so may also be implemented in an environment that lacks a GUI, s
example of such an implementation shown in FIG. 13, the selectors 56, 58 are words
character-based border identifying them as selectors. The user then selects a selecto
above.

In another example of a character-based user interface 50, the selectors 56, 58 are r
window. Function keys 270 on the keyboard 110 are implemented in place of the sel
according to the labels indicated at 290.

It is to be understood that this specification is provided by way of illustration and tha
define the invention.

<- Previous Patent (Garbage collection in log-structured inf..) | Next F public..) ->

W3C Architecture domain

SGML, XML, and Structured Document Interchange

This is W3C activity statement for SGML, XML, and Structured Document Interchange. It is one of the Architecture Domain activities. (See also the XML Overview for resources such as news, events, and history, background, and technical specifications.)

Jon Bosak, SGML WG chairman
Dan Connolly, SGML activity contact

Last revised $Date: 2000/06/26 13:49:43 $
created January 1996

Introduction

Most documents on the Web are stored and transmitted in HTML. HTML is a simple language well-suited for hypertext, multimedia, and the display of small and reasonably simple documents. HTML is based on SGML (Standard Generalized Markup Language), an ISO standard system for defining and using document formats.

SGML makes it possible to define your own formats for your own documents, to handle large and complex documents, and to manage large information repositories. Allowing generic SGML in Web documents would facilitate large-scale commercial Web publishing and make it much easier to apply advanced technologies such as Java to Web documents on the user's desktop. However, the full SGML specification is very expensive to implement and goes beyond the needs of the great majority of Web users.

The XML activity is dedicated to bringing the key benefits of generic SGML to the Web in a manner that is easy to implement and understand while remaining fully compliant with the ISO standard.

The goal of the activity is to enable an ISO-compliant subset of SGML, the Extensible Markup Lanaguage (XML), to be served, received, and processed on the Web. As in the case of HTML, the implementation of XML on the Web will require attention not just to structure and content, but also to the standardization of linking and display functions. Since a key design feature of XML is its clear separation of syntax from other processing behaviors, the explicit standardization of the most important of those behaviors (linking and stylesheets) is a necessary part of the XML activity in order to ensure the

vendor- and platform-neutral interoperation of XML documents. As in the case of XML syntax, the standardization of XML linking and stylesheets takes place within the context of existing international text processing standards.

Requirements

Web servers and clients conforming to the relevant standards must be able to exchange generic XML documents in a transparent manner. In particular:

- Web servers with XML content must be able prepare data for transmission. This typically includes the generation of a context wrapper with each XML fragment together with pointers to an associated Document Type Definition (DTD) and one or more stylesheets.
- Clients that process XML must be able to unpackage the fragment, parse it in context according to a DTD if required to do so by the document, render it (if a rendering application) in accordance with a specified stylesheet, and correctly interpret hypertext semantics (links, etc.) associated with various document elements.

Products

The specific deliverables of the SGML WG are being developed in three phases, as follows:

- **Phase I: A specification for the syntax of XML (Extensible Markup Language), a simplified version of SGML (ISO 8879) suitable for Internet applications.** The latest working draft of WD-xml-lang was released in March 1997; this is also available in a compressed PostScript version. A Japanese translation of WD-xml-lang has been made by Fuji Xerox. The next revision of WD-xml-lang is scheduled to be released June 30, 1997.
- **Phase II: A specification of standard hypertext mechanisms for XML applications based on HyTime (ISO/IEC 10744) and the Guidelines of the Text Encoding Initiative.** An initial draft of the xml-link spec was presented at the WWW6 Conference (Santa Clara, April 1997). This is also available in a compressed PostScript version. A Japanese translation of WD-xml-link has been made by Fuji Xerox. The next draft revision of WD-xml-link is scheduled to be released June 30, 1997.
- **Phase III: The specification of a standard stylesheet language for XML publishing applications based on DSSSL (ISO/IEC 10179)** together with public text and extensions needed to apply the DSSSL stylesheet language to Web browsers. Target delivery: a draft (WD-xml-style) to be delivered at the SGML/XML 97 Conference in Washington, D. C., December 1997.

Current situation

A paper titled "XML, Java, and the Future of the Web" gives one view of the kind of advanced Web applications made possible by XML; this paper is also available in a compressed PostScript version that demonstrates the application of DSSSL.

Accomplishments of this activity so far include:

- Formation of an SGML Working Group that coordinates with existing related standards efforts and provides specifications where needed to form a complete SGML Internet solution.
- Delivery of a public report on the generic SGML activity within the W3C at the Seybold Conference in San Francisco, September 1996.

- Completion of the first working draft of the syntactic component of XML, a subset of SGML designed for Internet applications.
- Presentation of the XML draft at the SGML 96 Conference in Boston, November 1996.
- Formation of the xml-dev mailing list for XML developers hosted by Imperial College, London.
- Creation of an XML FAQ maintained by Peter Flynn.
- Presentation by the Graphic Communications Association of the first XML Conference in San Diego, March 1997.
- Publication of the revised version of Part 1: Syntax in March 1997.
- Publication of the first draft of Part 2: Linking in April 1997 at the WWW6 Conference in Santa Clara.
- Approval of a Technical Corrigendum to ISO 8879:1986 to align features of XML with the SGML standard at the May 1997 meeting of ISO/IEC JTC1/SC18/WG8 in Barcelona.

Next step

Activities within the W3C

The SGML ERB and WG are currently completing Phases I and II of the XML activity described above.

Requirements for certain enhancements to XML syntax (structured attributes, alternative schemas, multiple name spaces) requested by other W3C activities are under currently under review, and these features may become part of the XML 1.0 working draft before it is submitted for approval as a W3C recommendation.

On July 1, 1997, the group currently known as the W3C SGML Editorial Review Board will become known as the W3C XML Working Group and will begin working under the process rules currently governing the activities of W3C working groups. On the same date, the group currently known as the W3C SGML Working Group will become known as the W3C XML Interest Group and will begin working under the process rules currently governing the activities of W3C interest groups.

Activities outside the W3C

Related efforts are currently taking place outside the W3C:

- Work on transmission of SGML fragments is well advanced within a technical committee of SGML Open, the consortium for SGML tools vendors.
- SGML Open has also specified a catalog mechanism for managing SGML document entities.
- ISO/IEC JTC1/SC18/WG8 has approved a Technical Corrigendum to ISO 8879:1986 in order to reconcile certain user requirements of XML with the SGML standard. That corrigendum has been forwarded to ISO for formal balloting.

- Enter your e-mail address to receive e-mail (courtesy of NetMind) when this page is updated:
  [ ] Register

DSSSL is an acronym for ...

Acronym Finder

DSSSL

DSSSL is an acronym for ...

* Document Style Semantic and Specification Language

More information about the definition of DSSSL may appear below:

Document Style Semantics and Specification Language (ISO/IEC 10179:1996). An internat stylesheet language for SGML/XML documents.

DSSSL stands for Document Style Semantics and Specification Language. It's an ISO stanc 10179:1996). The DSSSL standard is internationally used as a language for documents sty for SGML.

(Document Style Semantics and Specification Language) This draft international standard (applies to the specification of processing information for SGML documents. DSSSL is expec an international standard.

Document Style Semantics and Specification Language, A stylesheet language for SGML ar documents. A subset of DSSSL (DSSSL-O) is implemented by James Clark's Jade. A freewa checker for DSSSL is Henry Thompson's DSC. James Clark's DSSSL Page, OpenJade, the s DSSSL user group, A list of DSSSL Tutorials by Frank Boumphrey, The DSSSL Documentat collective effort to write a DSSSL documentation, hosting among others the 'DSSSL Handb 'DSSSL Cookbook', The DSSSList Homepage.

Document Style and Semantics Specification Language. The international standard for styl SGML documents. (ISO 10179)

Document Style Semantics Specification Language

Document Style Semantics and Specification Language (See definition)

Document Style Semantics and Specification Language (DSSSL) is a standard for the proc Standard Generalized Markup Language documents. DSSSL describes how such a structure might be presented visually, or converted to something else, or processed in some other w document structure language; DSSSL is a document processing language, especially for pr transformation. DSSL contains separate parts and you can choose which parts of the stand creating a DSSSL definition. It contains standards for a style language, Flow objects, a trai language, a document model, and a query language. Introduction to DSSSL Jade - James'

Document Style Semantics and Specification Language (ISO/IEC 10179:1996). An internat stylesheet language for SGML/XML documents.

Document Semantic & Style Specification Language

Document Style Semantics and Specification Language. DSSSL provides formatting inform. documents.

Document Style Semantics and Specification Language. The standard language for process converting documents, DSSSL can be used to translate HTML to AML.

Document Style Semantics and Specification Language. ISO/IEC DIS 10179:1990 The obje DSSSL Standard is to provide a formal and rigorous means of expressing the range of doci production specifications, including high-quality typography, required by the graphics arts net yet a full International Standard.

Document Style Semantics and Specification Language

Document Style Semantics and Specification Language (ISO/IEC 10179:1996). DSSSL styl used to render XML or SGML documents.

A superset of XSL. DSSSL is a document style language used primarily with SGML (Standa Markup Language) files.

A general purpose system of style sheets for SGML.

Document Style Semantics and Specification Language (See http://www.w3.org/Style/)

Acronym for "Document Style Semantics and Specification Language" which is the Styling SGML. In similar fashion to SGML it is very general, very powerful and very difficult to use. to SGML roughly as XSL relates to XML and as CSS relates to HTML.

The original Standard Generalized Markup Language (SGML) style sheet language. It is an and an established mechanism for directing the display of documents that are described b markup languages.

Every attempt has been made to provide you with the correct acronym for DSSSL. If we m we would greatly appreciate your help by entering the correct or alternate meaning in the

Definitions have been compiled from popular search engines and multiple results provided

---

WD-xml-961114

Extensible Markup Language (XML)

W3C Working Draft 14-Nov-96

This version:
http://www.w3.org/pub/WWW/TR/WD-xml-961114.html
Previous versions:
Latest version:
http://www.textuality.com/sgml-erb/WD-xml.html

Editors:
Tim Bray (Textuality) <tbray@textuality.com>
C. M. Sperberg-McQueen (University of Illinois at Chicago) <cmsmcq@uic.edu>

Status of this memo

This is a W3C Working Draft for review by W3C members and other interested parties. It is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work in progress". A list of current W3C working drafts can be found at: http://www.w3.org/pub/WWW/TR

Note: since working drafts are subject to frequent change, you are advised to reference the above URL, rather than the URLs for working drafts themselves.

This work is part of the W3C SGML Activity.

Abstract

Extensible Markup Language (XML) is an extremely simple dialect of SGML which is completely described in this document. The goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. For this reason, XML has been designed for ease of implementation, and for interoperability with both SGML and HTML.
Note on status of this document: This is even more of a moving target than the typical W3C working draft. Several important decisions on the details of XML are still outstanding - members of the W3C SGML Working Group will recognize these areas of particular volatility in the spec, but those who are not intimately familiar with the deliberative process should be careful to avoid actions based on the content of this document, until the notice you are now reading has been removed.

Table of Contents

---

1. Introduction

Extensible Markup Language, abbreviated XML, describes a class of data objects stored on computers and partially describes the behavior of programs which process these objects. Such objects are called XML documents. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language [ISO 8879].

XML documents are made up of storage units called entities, which contain either text or binary data. Text is made up of characters, some of which form the character data in the document, and some of which form markup. Markup encodes a description of the document's storage layout, structure, and arbitrary attribute-value pairs associated with that structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.

A software module called an *XML processor* is used to read XML documents and provide access to their content and structure. It is assumed that an XML processor is doing its work on behalf of another module, referred to as the *application*. This specification describes some of the required behavior of an XML processor in terms of the manner it must read XML data, and the information it must provide to the application.

1.1 Origin and Goals

XML was developed by a Generic SGML Editorial Review Board formed under the auspices of the W3 Consortium in 1996 and chaired by Jon Bosak of Sun Microsystems, with the very active participation of a Generic SGML Working Group also organized by the W3C. The membership of these groups is given in an appendix.

The design goals for XML are:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness is of minimal importance.

This specification, together with the associated standards, provides all the information necessary to understand XML version 1.0 and construct computer programs to process it.

This version of the XML specification (0.01) is for internal discussion within the SGML ERB only. It should not be distributed outside the ERB.

Known problems in version 0.01:

1. Several items in the bibliography have no references to them; several references in the text do not point to anything in the bibliograpy.
2. The EBNF grammar has not been checked for completeness, and has at least two start productions.
3. The description of conformance in the final section is incomplete.
4. Language exists in the spec which describes the effect of several decisions which have not been taken. Specifically, XML may have INCLUDE/IGNORE marked sections as does SGML, the comment syntax may change, XML may have CONREF attributes, the 8879 syntax for EMPTY elements may be outlawed, XML may choose to rule out what 8879 calls "ambiguous" content models, XML may choose to prohibit overlap between enumerated attribute values for different

attributes, the handling for attribute values in the absence of a DTD may be specified, there may be a way to signal whether the DTD is complete, the DTD summary may be dropped, and XML may support parameter entities, and XML may predefine a large number of character entities, for example those from HTML 3.2.

1.2 Relationship to Other Standards

Other standards relevant to users and implementors of XML include:

1. SGML (ISO 8879-1986). Valid XML Documents are SGML documents in the sense described in ISO standard 8879.
2. Unicode, ISO 10646. This specification depends on ISO standard 10646 and the technically identical Unicode Standard, Version 2.0, which define the encodings and meanings of the characters which make up XML text data.
3. IETF RFC 1738. This specification defines the syntax and semantics of Uniform Resource Locators, or URLs.
4. World-Wide Web Consortium Working Draft WD-html32-960909: HTML 3.2 Reference Specification. This includes the repertoire of characters to be predefined by an XML processor.

1.3 Notation

The formal grammar of XML is given using a simple Extended Backus-Naur Form (EBNF) notation. Each rule in the grammar defines one non-terminal or terminal symbol of the grammar, in the form

```
symbol ::= expression
```

Symbols are written with an initial capital letter if they are defined by a regular expression, with an initial lowercase letter if they have a more complex definition (i.e. if they require a stack for proper recognition). Literal strings are quoted; unless otherwise noted they are not case-sensitive. The distinction between symbols which can and cannot be recognized using simple regular-expressions is made for clarity only. It may be reflected in the boundary between an implementation's lexical scanner and its parser, but there is are no assumptions about the placement of such a boundary, nor even that the implementation has separate modules for parser and lexical scanner.

Within the expression on the right-hand side of a rule, the meaning of symbols is as shown below:

`#NN`
: where *NN* is a decimal integer, the expression matches the character in ISO 10646 whose UCS-4 bit-string, when interpreted as an unsigned binary number, has the value indicated

`#xNN`
: where *NN* is a hexadecimal integer, the expression matches the character in ISO 10646 whose UCS-4 bit-string, when interpreted as an unsigned binary number, has the value indicated

`[#xNN-#xNN], [a-zA-Z]`
: matches any character with a value in the range(s) indicated (inclusive)

`[^#xNN-#xNN], [^a-z]`
: matches any character with a value *outside* the range indicated

`[^abc]`
: matches any character with a value not among the characters given

`"string"`

matches the literal string given inside the double quotes

`'string'`
: matches the literal string given inside the single quotes

`a b`
: *a* followed by *b*

`a | b`
: *a* or *b* but not both

`a?`
: *a* or nothing; optional *a*

`a+`
: one or more occurrences of *a*

`a*`
: zero or more occurrences of *a*

`(expression)`
: *expression* is treated as a unit; allows subgroups to carry the operators `?`, `*`, or `+`

`/* ... */`
: comment

`[ WFC: ... ]`
: Well-formedness check; this identifies by name a check for well-formedness that is associated with a production.

`[ VC: ... ]`
: Validity check; this identifies by name a check for validity that is associated with a production.

1.4 Terminology

Some terms used with special meaning in this specification are:

`may`
: Conforming data and software may but need not behave as described.

`must`
: Conforming data and software must behave as described; otherwise they are in error.

`error`
: A violation of the rules of this specification; results are undefined. Conforming software may detect and report an error and may recover from it.

`reportable error`
: An error which conforming software must report to the user, unless the user has explicitly disabled error reporting.

`validity constraint`
: A rule which applies to all valid XML documents. Violations of validity constraints are errors; they must be reported by validating XML processors.

`well-formedness constraint`
: A rule which applies to all well-formed XML documents. Violations of well-formedness constraints are reportable errors.

`at user option`
: Conforming software may or must (depending on the verb in the sentence) provide users a means to select the behavior described; it must also allow the user *not* to select it.

`match`
: Case-insensitive match: two strings or names being compared must be identical except for differences between upper- and lower-case letters in scripts which have such a distinction. Characters with multiple possible representations in ISO 10646 (e.g. both precomposed and

base+diacritic forms) match only if they have the same representation, except for case differences, in both strings. Case folding must be performed as specified in The Unicode Standard, Version 2.0, section 4.1; in particular, it is recommended that case-insensitive matching be performed by folding uppercase letters to lowercase, not vice versa.

`exact(ly) match`
: Case-sensitive match: two strings or names being compared must be identical. Characters with multiple possible representations in ISO 10646 (e.g. both precomposed and base+diacritic forms) match only if they have the same representation in both strings.

`for compatibility`
: A feature of XML included solely to ensure that XML remains compatible with SGML; the expectation is that in many cases, those aspects of SGML that are not required to satisfy XML's requirements but mandated only to achieve conformance may be removed or replaced in the near future by the organizations that maintain that standard.

1.5 Common Syntactic Constructs

This section defines some symbols used widely in the grammar.

S (white space) consists of one or more blank characters, carriage returns, line feeds, or tabs.

< 1 White space >

```
S ::= (#x0020 | #x000a | #x000d | #x0009)+
```

For some purposes, characters are classified as letters, digits, or other characters:

< 2 Name >

```
Character ::= [#x20-              /* any ISO 10646 32-bit code */
              #xFFFFFFFF]
   Letter ::= [#x41-#x5A] |       /* Latin 1 upper and lowercase */
              [#x61-#x7A]
              | #xAA | #xB5
              | #xBA
              | [#xC0-#xD6]       /* Latin 1 supplementary letters */
              | [#xD8-#xF6]
              | [#xF8-#xFF]       /* Latin 1 supplementary letters */
              | [#x0100-          /* Extended Latin-A */
              #x017F]
              | [#x0180-          /* Extended Latin-B */
              #x0217]
              | [#x0250-          /* IPA extensions, spacing modifiers,
              #x1FFF]                diacritics, Greek, Coptic, Cyrillic,
                                     Armenian, Hebrew, ... */
              | [#x3040-          /* CJK */
              #x9FFF]
```

```
             | [#xF900-   /* CJK compatibility ideographs ... */
             #xFDFF]
             | [#xFE70-   /* Arabic presentation forms B */
             #xFEFE]
             | [#xFF21-   /* Fullwidth Latin A-Z */
             #xFF3A]
             | [#xFF41-   /* Fullwidth Latin a-z */
             #xFF5A]
             | [#xFF66-   /* Halfwidth katakana, hangul */
             #xFFDC]
Digit ::=    [#x0030-     /* Correct this table using section 4.5
             #x0039]         of Unicode 2.0
                             ISO 646 digits */
             | [#x0660-   /* Arabic-Indic digits */
             #x0669]
             | [#x06F0-   /* Eastern Arabic-Indic digits */
             #x06F9]
             | [#x0966-   /* Devanagari digits */
             #x096F]
             | [#x09E6-   /* Bengali digits */
             #x09EF]
             | [#x0A66-   /* Gurmukhi digits */
             #x0A6F]
             | [#x0AE6-   /* Gujarati digits */
             #x0AEF]
             | [#x0B66-   /* Oriya digits */
             #x0B6F]
             | [#x0BE7-   /* Tamil digits (no zero) */
             #x0BEF]
             | [#x0C66-   /* Telugu digits */
             #x0C6F]
             | [#x0CE6-   /* Kannada digits */
             #x0CEF]
             | [#x0D66-   /* Malayalam digits */
             #x0D6F]
             | [#x0E50-   /* Thai digits */
             #x0E59]
             | [#x0ED0-   /* Lao digits */
             #x0ED9]
             | [#xFF10-   /* Fullwidth digits
             #xFF19]         Ranges taken from Java documentation.
                             Check against Unicode 2.0, section
                             4.6.
                             N.B. not clear whether the relevant
                             Greek and Hebrew letters should also
```

```
be digits. Will matter for NUMBER
attributes. */
```

A *Name* is a token beginning with a letter or hyphen and continuing with letters, digits, hyphens, or full stops (together known as name characters). The use of any name beginning with a string which matches `"-XML-"` in a fashion other than those described in this specification is a reportable error.

A *Number* is a sequence of digits. An *Nmtoken* (name token) is any mixture of name characters.

< 3 Names, Numbers, and Tokens >

```
    Name ::= (Letter | '-') (Letter | Digit | '-' | '.')*
  Number ::= Digit+
 Nmtoken ::= (Letter | Digit | '-' | '.')+
Nmtokens ::= Nmtoken (S Nmtoken)*
```

Literal data is any quoted string containing neither the quotation mark used as a delimiter nor angle brackets. It may contain entity and character references.

< 4 Literals >

```
    Literal ::= '"' [^"<>]* '"'
              | "'" [^'<>]* "'"
QuotedCData ::= '"' [^"<>]* '"'
              | "'" [^'<>]* "'"
QuotedNames ::= '"' Names '"' | "'" Names "'"
```

2. Documents

A textual object is an *XML Document* if it is either valid, or failing that, well-formed, as defined in this specification.

2.1 Logical and Physical Structure

Each XML document has both a logical and a physical structure.

Physically, the document is composed of units called entities; it begins in a "root" or document entity, which may refer to other entities, and so on ad infinitum. Entities referred to are embedded in the document at the point of reference.

The document contains declarations, elements, comments, entity references, character references, and processing instructions, all of which are indicated in the document by explicit markup. These concepts and their markup are all explained elsewhere in this specification.

The two structures must be synchronous: tags and elements must each begin and end in the same entity,

but may refer to other entities internally; comments, processing instructions, character references, and entity references must each be contained entirely within a single entity. Entities must each contain an integral number of elements, comments, processing instructions, and references, possibly together with character data not contained within any element in the entity, or else they must contain non-textual data, which by definition contains no elements.

2.2 Characters

The data stored in an XML entity is either text or binary. Binary data has an associated notation, identified by name; beyond a requirement to make available the notation name and the size in bytes of the binary data in a storage object, XML provides no information about and places no constraints on binary data. In fact, so-called binary data may in fact be textual, perhaps even well-formed XML text; but its identification as binary means that an XML processor need not parse it in the fashion described by the specification. XML text data is a sequence of characters. A character is A character is an atomic unit of text represented by a bit string; valid bit strings and their meanings are specified by ISO 10646.

Users may extend the ISO 10646 character repertoire, in the rare cases where this is necessary, by making use of the private use areas.

The mechanism for encoding character values into bit patterns may vary from entity to entity. All XML processors must accept the UTF-8 and UCS-2 encodings of 10646; the mechanisms for signalling which of the two are in use, and for bringing other encodings into play, are discussed later, in the discussion of character encodings.

Regardless of the specific encoding used, any character in the ISO 10646 character set may be referred to by the decimal or hexadecimal equivalent of its bit string:

< 5 Character references >

```
    Hex ::= [0-9a-fA-F]
   Hex4 ::= Hex Hex Hex Hex
CharRef ::= '&#' Number ';'
          | '&u-' Hex4 ';'
```

2.3 Syntax of Text and Markup

XML text consists of intermingled character data and markup. Markup takes the form of start-tags, end-tags, empty elements, entity references, character references, comments, marked sections, document type declarations, and processing instructions. The simplest form of XML processor thus could parse a well-formed XML document using the following rules:

< 6 Trivial text grammar >

```
Trivial ::= (PCDATA | Markup)*
     Eq ::= S? '=' S?
 Markup ::= '<' Name (S Name Eq QuotedCData)*   /* start-tags */
            S? '>'
```

```
| '</' Name S? '>'                         /* end-tags */
| '<' Name (S Name Eq QuotedCData)         /* empty elements */
* S? '/>'
| '&' Name ';'                             /* entity references */
| '&#' Number ';'                          /* character references
                                              */
| '&u-' Hex4 ';'                           /* character references
                                              */
| '<!--' [^-]* ('-' [^-]+)* '-->'          /* comments */
| '<![CDATA[' MsData ']]>'                 /* marked sections */
| '<!DOCTYPE' (Name | S)+                  /* doc type declaration
('[' [^]]* ']')? '>'                          */
| '<?' [^?]* ('?' [^>]+)* '?>'             /* processing
                                              instructions */
```

Most processors will require the more complex grammar given in the rest of this specification.

All text that is not markup constitutes the character data of the document.

The ampersand character (&) and the left and right angle bracket (< and >) may appear in their literal form *only* when used as markup delimiters, or within comments, processing instructions, or marked sections. If they are needed in the text, they must be represented using the strings "`&`", "`<`", and "`>`". Parsed character data is thus any string of characters which does not contain the start-delimiter of any markup. Character data is any string of characters not including the marked-section-close delimiter, "`]]>`". For convenience, the single-quote character (') may be represented as "`&sq;`", and the double-quote character (") as "`&dq;`".

< 7 Character Data >

```
PCDATA ::= [^<&]*
MsData ::= [^]]* ((('']' ([^]])) | (']]' [^>])) [^]]*)*
```

2.4 Comments

Comments may appear anywhere that character data may, except in a marked section (more properly, comments appearing in a marked section will not be recognized as such). They are not part of the document's character data; an XML processor may, but need not, make it possible for an application to retrieve the text of comments. An XML processor must inform the application of the length of comments if they are not passed through, to enable the application to keep track of the correct location of objects in the XML document. For compatibility, the string -- (double-hyphen) may not occur within comments.

< 8 Comments >

```
Comment ::= '<!--' [^-]* ('-' [^-]+)* '-->'
```

2.5 Processing Instructions

Processing instructions, usually referred to as PIs, allow the XML processor to pass instructions directly to selected applications.

< 9 Processing Instructions >

```
PI ::= '<?' Name S [^?]* ('?' [^>]+)* '?>'
```

PIs are not part of the document's character data, but must be passed through to the application. The Name which follows the '?' at the beginning of the PI is called the *PI target*. It is normally the name of a declared notation, identifying the application to which it belongs. The use of the PI target "XML" in any other way other than those described in this specification is a reportable error.

2.6 Marked Sections

Marked sections can occur anywhere character data may occur; they are used to escape blocks of text which may contain characters which would otherwise be recognized as markup. Marked sections begin with the string `<![CDATA[` and end with the string `]]>`:

< 10 Marked Sections >

```
      MS ::= MsStart MsData MsEnd
MsStart ::= '<![CDATA['
  MsEnd ::= ']]>'
```

Within a marked section, only the *MsEnd* string is recognized, so that angle brackets and ampersands may occur in their literal form and need not be escaped using `<`, etc. Marked sections cannot nest.

2.7 White Space Handling

While authoring XML documents, it is often convenient to use "white space" (spaces, tabs, blank lines, denoted by the nonterminal `S` in this specification) to set apart the markup for greater readability. Such white space is typically not intended for inclusion in the delivered version of the document. On the other hand, "significant" white space, to be retained in the delivered version, is common; for example in poetry or source code.

An XML processor must provide two distinct white space handling modes, *COLLAPSE* and *KEEP*, and have the ability to apply these modes on a per-element basis. They operate as follows:

`COLLAPSE`
: The XML processor must suppress (i.e. not pass to the application) all white space in an element which immediately follows the start-tag and all that which immediately precedes the end-tag. In the element's character data, it must convert all sequences of white space characters to a single space (#x0020) character, before passing the data to the application.

`KEEP`
: The XML processor must suppress initial line break characters which immediately follow the

start-tag of the element, and which immediately precede its end-tag. All other characters in the character data of the element must be passed to the application without change.

The white space handling mode is signaled through the use of a reserved attribute, whose declaration is as follows:

```
<!ATTLIST * -XML-SPACE (KEEP|COLLAPSE) #IMPLIED>
```

where the "*" signifies that this attribute may apply to any element.

The value of the attribute sets the white space handling mode for the element and for any contained elements. Unless otherwise specified, an XML processor is to set the white space handling mode for the root element of a document to *COLLAPSE*.

2.8 Prolog and Document Type Declaration

The function of the markup in an XML document is to describe its storage and logical structures, and associate attribute-value pairs with the logical structure. XML provides a mechanism, the document type declaration, to define constraints on that logical structure, and to support the use of predefined storage units. An XML document is said to be *valid* if there is an associated document type declaration and if the document complies with the constraints expressed in it.

The document type declaration must appear before the first element in the document.

< 11 XML document >

```
document ::= Prolog element Misc*
  Prolog ::= EncodingDecl? Misc* RMDecl? Misc* (doctypdecl |
             DtdSummary)? Misc*
    Misc ::= Comment | PI | S
```

For example, the following is a complete XML document, well-formed but not valid:

```
<greeting>Hello, world!</greeting>
```

The XML *document type declaration* may include a pointer to an external entity containing a subset of the necessary markup declarations, and may also directly include another, internal, subset of the necessary markup declarations.

< 12 Document type declaration >

```
   doctypedecl ::= '<!DOCTYPE' S Name Extid? S? ('[' internalsubset*
                   ']' S?)? '>'
internalsubset ::= elementdecl | AttlistDecl | EntityDecl
                   | NotationDecl | DtdSummary | S | Comment
```

These two subsets, taken together, are properly referred to as the *document type definition*, abbreviated *DTD*. However, it is a common practice for the bulk of the markup declarations to appear in the external subset, and for this subset, usually contained in a file, to be referred to as "the DTD" for a class of documents. For example:

```
<!DOCTYPE greeting SYSTEM "hello.dtd">
<greeting>Hello, world!</greeting>
```

The system identifier `hello.dtd` indicates the location of a full DTD for the document.

The declarations can also be given locally, as in this slightly larger example:

```
<?XML encoding="UTF-8">
<!DOCTYPE greeting [
<!ELEMENT greeting (#PCDATA)>
]>
<greeting>Hello, world!</greeting>
```

The character-set label `<?XML encoding="UTF-8">` indicates that the document entity is encoded using the UTF-8 transformation of ISO 10646. The legal values of the character set code are given in the discussion of character encodings.

Individual markup declaration types are described elsewhere in this specification.

2.9 Required Markup Declaration

In many cases, an XML processor can read an XML document and accomplish useful tasks without having first processed the entire DTD. However, certain declarations can substantially affect the actions of an XML processor. A document author can communicate whether or not DTD processing is necessary using a *required markup declaration* (abbreviated RMD) processing instruction:

< 13 Required markup declaration >

```
RMDecl ::= '<?XML' S 'RMD' Eq ('NONE' | 'INTERNAL' | 'ALL') S? '?>'
```

In an RMD, the value *NONE* indicates that an XML processor can parse the containing document correctly without first reading any part of the DTD. The value *INTERNAL* indicates that the XML processor must read and process the internal subset of the DTD to parse the containing document correctly. The value *ALL* indicates that the XML processor must read and process the declarations in both the subsets of the DTD to parse the containing document directly.

The RMD must indicate that the entire DTD is required if the external subset contains any declarations of

1. undistinguished empty elements, and these elements occur in the document, or
2. attributes with default values, and elements to which these attributes apply appear in the document, or
3. entities, and references to those entities appear in the document.

If such declarations occur in the internal but not the external subset, the RMD should take the value *INTERNAL*. It is an error to specify *INTERNAL* if the external subset is required, or to specify *NONE* if the internal or external subset is required.

If no RMD is provided, the effect is identical to an RMD with the value *ALL*.

3. Logical Structures

Each XML document contains one or more *elements*, the boundaries of which are either delineated by start-tags and end-tags, or, for empty elements, are limited to the start-tag. Each element has a type, identified by name (sometimes called its *generic identifier* or *GI*), and may have a set of attributes. Each attribute has a name and a value.

This specification does not constrain the semantics, use, or (beyond syntax) names of the elements and attributes.

3.1 Start- and End-Tags

The beginning of every XML element is marked by a *start-tag*.

< 14 Start-tag Recognition >

```
     STag ::= '<' Name (S Attribute)* S?
              '>'
Attribute ::= Name Eq QuotedCData              [ VC: Attribute Value
                                                 Type ]
```

The *Name* in the start- and end-tag rules gives the element's *type*. The *Name-QuotedCData* pairs are referred to as the *attributes* of the element, with the *Name* referred to as the *attribute name* and the content of the *QuotedCData* (the characters between the "'" or '"' delimiters) as the *attribute value*.

Validity Constraint - Attribute Value Type:
Attribute values must be of the type declared for the attribute. (For attribute types, see the discussion of attribute declarations.)

The end of every element which is not empty is marked by an *end-tag*:

< 15 End-tag Recognition >

```
ETag ::= '</' Name S? '>'
```

The Name, once again, gives the element's type.

If an element is *empty*, the start-tag constitutes the whole element. An empty element takes a special form:

< 16 Tags for empty elements >

```
EmptyElement ::= '<' Name (S Attribute)* S? '/>';
```

For compatibility, an empty element may have the same syntax as a start-tag; in this case, it cannot be recognized based on syntax, but must be declared as being empty. Such elements are called *undistinguished empty elements*.

The text between the start-tag and end-tag is called the element's *content*:

< 17 Content of elements >

```
content ::= (element | PCDATA | MS | PI | Comment)
            *
element ::= EmptyElement                             /* empty elements
                                                        */
            | STag content ETag                      [ WFC: GI Match ]
```

Well-Formedness Constraint - GI Match:
The *Name* in an element's end-tag must match that in the start-tag.

3.2 Well-Formed XML Documents

A textual object is said to be a *well-formed* XML document if, first, it matches the production above labeled *XML Document*, and if:

1. There are no undistinguished empty elements which have not been specified as such in an element declaration.
2. For each entity reference which appears in the document, the entity name has been declared in the document type declaration.

Matching the "XML Document" production implies that:

1. It contains one or more elements.
2. There is one element, called the *root*, for which neither the start-tag nor the end-tag are in the content of any other element. For all other elements, if the start-tag is in the content of another element, the end-tag is in the content of the same element. More simply stated, the elements, delineated by start- and end-tags, nest within each other properly.

As a consequence of this, for each non-root element *C*, there is one other element *P* such that *C* is in the content of *P*, but is not in the content of any other element that is in the content of *P*. Then *P* is referred to as the *parent* of *C*, and *C* as the *child* of *P*.

3.3 Element Declaration

The element structure of an XML document may be declared fully or partially. Such declarations serve two purposes:

1. To establish a set of structural constraints, i.e. a grammar, for a class of documents, and to verify that documents are valid, i.e. comply with that grammar.
2. To make XML documents well-formed by declaring undistinguished empty elements.

An element declaration constrains the element's type and its content. The content constraints will be described first; four forms are available: empty, any, mixed content, and element content.

Declarations often contain references to element types, for example when constraining which element types can appear as children of others, and which attributes may be attached to which element types. At user option, an XML processor may issue a warning when no element is declared whose type matches that given, but this is not an error.

An *element declaration* takes the form:

< 18 Element declarations >

```
elementdecl ::= '<!ELEMENT' S Name S ('EMPTY' |      [VC: Unique Element
                'ANY' | Mixed | elements ) S?         Declaration ]
                '>'
```

where *Name* identifies the type of the element.

Validity Constraint - Unique Element Declaration:
No element type may be declared more than once.

For compatibility, an element type may be declared using the keyword EMPTY; this is an undistinguished empty element.

If an element type is declared using the keyword ANY, then there are no constraints on its content aside from its being well-formed: it may contain subelements of any type and number, interspersed with character data.

3.3.1 Mixed Content

An element type may be declared to contain mixed content, that is text comprising character data optionally interspersed with child elements. In this case, the types of the child elements are constrained, but not their order nor their number of occurrences:

< 19 Mixed-content declaration >

```
Mixed ::= '(' S? '#PCDATA' ( S? '|' S? Name )* S? ')*'
        | '(' S? '#PCDATA' S? ')'
```

where the *Names* give the types of elements that may appear as children.

3.3.2 Element Content

An element type may be declared to have element content, consisting only of other elements. In this

case, the constraint includes a content model, a context-free grammar governing the allowed types of the child elements and the order in which they appear. The grammar is built on content particles (CPs), which consist of names, choice lists of content particles, or sequence lists of content particles:

< 20 Element-content models >

```
elements ::= cp
      cp ::= (Name | choice | seq) ('?' | '*' | '+')?
     cps ::= S? cp S?
     seq ::= '(' cps (',' cps)* ')'
  choice ::= '(' cps ('|' cps)+ ')'
```

where each *Name* gives the type of an element which may appear as a Child. Any content particle in a choice list may appear in the element content at the appropriate location; content particles occurring in a sequence list must each appear in the element content in the order given. The optional character following a name or list governs whether the element or the content particles in the list may occur one or more, zero or more, or zero or one times respectively. The syntax and meaning is identical to that used in the productions in this specification.

3.4 Attribute Declaration

Attributes are used to associate name-value pairs with elements. Attributes may appear only within start-tags; thus, the productions used to recognize them appear in that discussion. Attribute declarations may be used:

1. To define the set of attributes pertaining to a given element type.
2. To establish a set of type constraints on these attributes.
3. To provide default values for atributes.

Attribute-list declarations specify the name, data type, and default value (if any) of each attribute associated with a given element type:

< 21 Attribute list declaration >

```
AttlistDecl ::= '<!ATTLIST' S Name         [ VC: Unique Attribute-List
                AttDef+ S? '>'               Declaration ]
     AttDef ::= S Name S AttType S         [ VC: Unique Attribute Name ]
                Default
```

The *Name* in the *AttlistDecl* rule is the type of an element; it is a reportable error to declare attributes for an element type not itself declared. The *Name* in the *AttDef* rule is the name of the attribute.

Validity Constraint - Unique Attribute-List Declaration:
At most one attribute-list declaration may be provided for a given element type.

Validity Constraint - Unique Attribute Name:
An attribute name may appear at most once in an attribute-list declaration.

3.4.1 Attribute Types

XML attribute types are of three kinds: a string type, a set of tokenized types, and enumerated types. The string type may take any literal string as a value; the tokenized types have varying lexical and semantic constraints, as noted:

< 22 Attribute types >

```
      AttType ::= StringType | TokenizedType |
                  EnumeratedType
   StringType ::= 'CDATA'
TokenizedType ::= 'ID'                           [ VC: ID ]
                | 'IDREF'                        [ VC: Idref ]
                | 'IDREFS'                       [ VC: Idref ]
                | 'ENTITY'                       [ VC:
                                                  EntityName ]
                | 'ENTITIES'                     [ VC:
                                                  EntityName ]
                | 'NMTOKEN'                      [ VC: Name
                                                  Token ]
                | 'NMTOKENS'                     [ VC: Name
                                                  Tokens ]
```

Validity Constraint - ID:
Values of this type must be valid Names. A name must not appear more than once in an XML document as a value of this type: i.e. ID values must uniquely identify the elements which bear them.

Validity Constraint - Idref:
Values of this type must be one (for IDREFS one or more) Names, which must each match the value of an ID attribute on some element in the XML document; i.e. IDREF values must match some ID.

Validity Constraint - Entity Name:
Values of this type must be one (for ENTITIES one or more) Names, which must each exactly match the name of an external binary entity declared in the XML DTD.

Validity Constraint - Name token:
Values of this type must consist of a string matching the `Nmtoken` nonterminal of the grammar defined in this specification. The XML processor must normalize the case of the attribute value before passing it to the application.

Validity Constraint - Name tokens:
Values of this type must consist of a string matching the `Nmtokens` nonterminal of the grammar defined in this specification. The XML processor must reduce white space to single blanks, and normalize the case of the attribute value, before passing it to the application.

Enumerated attributes can take one of a list of values provided in the declaration; there are two types:

< 23 Enumerated attribute declarations >

```
EnumeratedType ::= NotationType | Enumeration
  NotationType ::= 'NOTATION' S '(' S? Name (S?        [VC: Notation
                   '|' S? Name)* S? ')'                 Attributes ]
   Enumeration ::= '(' S? Nmtoken (S? '|' S?           [VC: Enumeration ]
                   Nmtoken)* S? ')'
```

Validity Constraint - Notation Attributes:
The names in the declaration of NOTATION attributes must be names of declared notations (see the discussion of notations). Values of this type must match one of the notation names included in the declaration.

Validity Constraint - Enumeration:
Values of this type must match one of the *Nmtoken* tokens in the declaration.

3.4.2 Attribute Defaults

An attribute declaration may provide information on whether the attribute's presence is required, and if not, how an XML processor should react if a declared attribute is absent in a document:

< 24 Attribute defaults >

```
Default ::= '#REQUIRED'
          | '#IMPLIED'
          | ( '#FIXED'? Literal)
```

#REQUIRED means that it is a reportable error should the processor encounter a start-tag where this attribute is omitted, i.e. could occur but does not. #IMPLIED means that if an attribute is omitted, the XML processor must inform the application that no value was specified; no constraint is placed on the behavior of the application.

If the attribute is neither #REQUIRED nor #IMPLIED, then the *Literal* value contains the declared *default* value. If the #FIXED is present, it is a reportable error if the attribute is present with a different value from the default. If a default value is declared, when an XML processor encounters an omitted attribute, it is to assume that the attribute is present and has the declared default value.

3.5 Partial DTD Information

The prolog of an XML document may contain an abbreviated summary of the DTD for the convenience of non-validating processors.

< 25 DTD summary >

```
DtdSummary ::= (EmptyInfo | TextInfo | NoText | IdInfo |
               DefaultInfo)*
```

```
  EmptyInfo ::= '<?XML' S 'empty' S 'names' Eq QuotedNames S? '?>'
   TextInfo ::= '<?XML' S 'text' S 'names' Eq QuotedNames S? '?>'
     NoText ::= '<?XML' S 'notext' S 'names' Eq QuotedNames S? '?>'
     IdInfo ::= '<?XML' S 'idinfo' S
                'ids' Eq QuotedPairs S
                'refs' Eq QuotedPairs S? '?>'
QuotedPairs ::= '"' Pairs '"' | "'" Pairs "'"
      Pairs ::= ('*' | Name) S Name (S ('*' | Name) S Name)*
DefaultInfo ::= '<?XML' S 'default' Name (S Name Eq Literal)* S? '?>'
```

The *empty*, *text*, and *notext* declarations give, respectively, lists of element types declared as empty, mixed-content, or element-content elements. The *idinfo* declaration lists attributes declared as id or idref (s) (in the *ids* and *refs* values, respectively). Each attribute name is given as an element-attribute pair; an asterisk matches all element types which have attributes of the given name. The *default* declaration specifies default values for the attributes of a given element type.

For example:

```
<?XML empty names="ptr xptr pb" ?>
<?XML notext names="div0 div1 div2 div3 list" ?>
<?XML text names="p emph q title hi" ?>
<?XML idinfo ids='* id' refs='* target * targets' ?>
<?XML default div type='section' ?>
```

Elements not listed as `notext` are implicitly declared as `text` elements; the *text* declaration is thus not strictly necessary.

4. Physical Structures

An XML document may consist of one or many storage units. The unit of storage is the *entity*; entities are identified by name and have *content*. Each XML document has one entity called the document entity, which serves as the starting point for the XML processor (and may contain the whole document).

Entitities may be either binary or text. A text entity contains text data which is to be considered as an integral part of the document. A binary entity contains binary data identified by notation. Text and binary entities cannot be distinguished by syntax; their types are established in their declarations, described below.

4.1 Character and Entity References

A *character reference* refers to a specific character in the ISO 10646 character set, often one not directly accessible from available input devices:

< 26 Character Reference >

```
CharRef ::= '&#' Number ';'
          | '&u-' Hex4 ';'
```

An *entity reference* refers to the content of a named entity.

< 27 Entity Reference >

```
Reference ::= EntityRef | CharRef
EntityRef ::= '&' Name ';'          [ WFC: Entity Declared ]
                                     [ WFC: Text Entity ]
                                     [ WFC: No Recursion ]
```

Well-Formedness Constraint - Entity Declared:
The *Name* given in the entity reference must match exactly the name given in the declaration of the entity.

Well-Formedness Constraint - Text Entity:
An entity reference may not contain the name of a binary entity. Binary entities may be referred to only in attribute values declared to be of type ENTITY or ENTITIES.

Well-Formedness Constraint - No Recursion:
A text entity may not contain a reference to itself.

4.2 Declaring Entities

Entities are declared thus:

< 28 Entity declaration >

```
EntityDecl ::= '<!ENTITY' S Name EntityDef S? '>'
 EntityDef ::= Literal | ExternalDef;
```

The *Name* is that by which the entity is invoked by exact match in an entity reference.

4.2.1 Internal Entities

If the entity definition is just a *Literal*, this is called an *internal* entity - there is no separate storage unit, and the content of the entity is given in the declaration. An internal entity is a text entity.

4.2.2 External Entities

If the entity is not internal, it is an *external entity*, declared as follows:

< 29 External entity declaration >

```
ExternalDef ::= NDataDecl? 'SYSTEM' Literal
  NDataDecl ::= 'NDATA' S Name S            [ VC: Notation Declared ]
```

If the *NDataDecl* is present, this is a binary data entity, otherwise a text entity.

Validity Constraint - Notation Declared:
The *Name* must match one of those in a notation declaration.

The *Literal* that follows the keyword SYSTEM called the entity's *system identifier*. It is a URL, which may be used to retrieve the content of the entity.

4.2.3 Character Encoding in Entities

Each text entity in an XML document may use a different encoding for its characters. All XML processors must be able to read entities in either the UTF-8 or UCS-2 encodings. It is recognized that for for some advanced work, particularly with Asian languages, the use of the UTF-16 encoding is required, and correct handling of this encoding is a desirable characteristic in XML processor implementations.

Entities encoded in UCS-2 must begin with the Byte Order Mark described by ISO 10646 Annex E and Unicode Appendix B (the ZERO WIDTH NO-BREAK SPACE character, U+FEFF) -- this is an encoding signature, not part of either the markup or character data of the XML document. XML processors must be able to use this character to differentiate between UTF-8 and UCS-2 encoded documents.

Although an XML processor is only required to read entities in the UTF-8 and UCS-2, it is recognized that many other encodings are in daily use around the world, and it may be advantageous for XML processors to read entities that use these non-standard encodings. For this purpose, XML provides an encoding declaration processing instruction, which must appear at the beginning of an entity, before any other character data or markup:

< 30 Encoding declaration >

```
Encodingdecl ::= '<?XML' S 'encoding' Eq QEncoding S? '?
                 >'
   QEncoding ::= '"' Encoding '"' | "'" Encoding "'"
    Encoding ::= 'UTF8' | 'UTF16' | 'UCS2' | 'UCS4'        /* Unicode */
                 | 'ISO8859' ('-' [1-9])?                   /* 8-bit */
                 | 'Shift-JIS' | 'EUC-JIS' | 'New-JIS'      /* Japanese
                                                              */
```

It is a reportable error for an entity including an encoding declaration to be stored in an encoding other than that named in the declaration.

An entity which begins with neither a Byte Order Mark nor an encoding declaration must be in the UTF-8 encoding.

While XML provides mechanisms for distinguishing encodings, it is recognized that in a heterogeneous

networked environment, there is often difficulty in reliably signaling the encoding of an entity. Errors in this area fall into two categories:

1. failing to read an entity because of inability to recognize its actual encoding, and
2. reading an entity incorrectly because of an incorrect guess of its proper encoding.

The first class of error is extremely damaging, and the second class is extremely unlikely. For these reasons, XML processors should make an effort to use all available information, internal and external, to aid in detecting an entity's correct encoding. Such information may include, but is not limited to:

1. Using information from an HTTP header
2. Using a MIME header obtained other than through HTTP
3. Metadata provided by the native OS filesystem or through other mechanisms
4. Analyzing the bit patterns at the front of an entity to determine if the application of any known encoding yields a valid encoding declaration.

If an XML processor encounters an entity with an encoding that it is unable to process, it must inform the application of this fact and allow the application to request either that the entity should be treated as an binary entity, or that processing should cease.

4.2.4 The Document Entity

The *document entity* serves as the root of the entity tree and a starting-point for an XML processor. This specification does not specify how the document entity is to be located by an XML processor; unlike other entities, the document entity might well appear on an input stream of the processor without any identification at all.

4.3 XML Processor Treatment of Entities

When an XML processor encounters a character reference or an entity name (in a reference or attribute value):

1. In all cases, the XML processor must inform the application of the reference's occurrence and its identifier (for an entity reference, the name; for a character reference, the "&..." string.)
2. For both character and entity references, the processor must remove the reference itself from the text data before passing the data to the application.
3. For character references, the processor must insert the indicated ISO 10646 bit pattern into the text data at the location of the reference before passing it to the application.
4. For an external entity reference, the processor must inform the application of the entity's system identifier.
5. If the external entity is binary, the processor must inform the application of the associated notation name, and the notation's associated system identifier.
6. For an internal (text) entity, the processor must *include* the entity; that is, retrieve its content, and treat the content as a part of the text data of the XML document, processing this replacement text data (which may contain both text and markup) to determine what data to pass to the application. Since the entity's content text may contain other entity references, an XML processor may have to repeat the inclusion process recursively.
7. If the entity is an external text entity, then in order to validate the XML document, the processor must include the content of the entity.
8. If the entity is an external text entity, and the processor is not attempting to validate the XML

document, the processor may, but need not, include the entity's content. This rule is based on the recognition that the inclusion semantic provided by the SGML and XML text entity mechanism, primarily designed to support modularity in authoring, may not be appropriate for other applications, in particular document browsing. Browsers, for example, when encountering an external text entity reference, might choose to provide a visual indication of the entity's presence and retrieve it for display only on demand. While this behavior would not allow the application to validate the document, it is compliant with this specification.

4.4 Notation Declaration

Notations identify by name the format of external binary entities.

Notation declarations provide a name for the notation, for use in entity and attribute declarations and in attribute-value specifications, and an external identifier for the notation which may allow an XML processor or its client application to locate a helper application capable of processing data in the given notation.

< 31 Notation declarations >

```
NotationDecl ::= '<!NOTATION' S Name S Extid S? '>'
```

XML processors must provide applications with the name and external identifier of any notation declared and referred to in an attribute value, attribute definition, or entity declaration. They may additionally resolve the external identifier into the system identifier, file name, API address, or other information needed to allow the application to call a processor for data in the notation described. (It is not an error, however, for XML documents to declare and refer to notations for which notation-specific applications are not available on the system where the XML processor or application is running.)

5. Conformance

Conforming XML processors fall into two classes: validating and non-validating.

Validating and non-validating systems alike must report violations of the well-formedness constraints given in this specification.

Validating systems must report locations in which the document does not comply with the constraints expressed by the declarations in the DTD. They must also report all failures to fulfill the validity constraints given in this specification.

A. XML and SGML

XML is designed to be a subset of SGML, in that every valid XML document should also be a valid SGML document, using the same DTD, and that the parse trees produced by an SGML parser and an XML processor should be the same. To achieve this, XML was defined by removing features and options from the specification of SGML.

- Despite this, there are a small number of cases where XML fails to be a pure subset of SGML, including:

  1. XML's white space handling rules are much less elaborate than those of SGML. One effect is that for elements using the KEEP mode for white-space handling, an XML processor will pass through a few record-separator markers that an SGML processor will eat.
  2. XML defines, for documents, the property of being well-formed; this does not really correspond to any SGML concept.

The following list describes features which are available in SGML but not in XML. It may not be complete.

1. Tag omission
2. The CONCUR, LINK, DATATAG, and SHORTREF features
3. The "&" connector in content models
4. Inclusions and exclusions in content models
5. CURRENT, CONREF, NAME, NAMES, NUMBER, NUMBERS, NUTOKEN, and NUTOKENS declarations for attributes
6. The NET construct
7. Abstract syntax
8. Capacities and quantities
9. Comments appearing within other markup declarations
10. Public Identifiers
11. Omission of quotes on attribute values

B. References

ISO/IEC 8879
: ISO (International Organization for Standardization). *ISO/IEC 8879-1986 (E). Information processing -- Text and Office Systems -- Standard Generalized Markup Language (SGML).* First edition -- 1986-10-15. [Geneva]: International Organization for Standardization, 1986.

ISO/IEC 10646
: ISO (International Organization for Standardization). *ISO/IEC 10646-1993 (E). Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane.* [Geneva]: International Organization for Standardization, 1993.

ISO/IEC 10744
: ISO (International Organization for Standardization). *ISO/IEC 10744-1992 (E). Information technology -- Hypermedia/Time-based Structuring Language (HyTime).* [Geneva]: International Organization for Standardization, 1992. *Extended Facilities Annexe.* [Geneva]: International Organization for Standardization, 1996. (?)

Unicode
: The Unicode Consortium. *The Unicode Standard: Version 2.0.* Reading, Mass.: Addison-Wesley Developers Press, 1996.

C. Working Group and Editorial Review Board Membership

C.1 Working Group

This specification was prepared with the assistance and participation of the W3C SGML Working

Group. Publication of this specification does not necessarily imply that all members of the working group agree with its content. At the time this specification was prepared, the W3C SGML Working Group comprised the following members.

1. Sharon Adler, EBT (sca@ebt.com)
2. Paula Angerstein, Texcel (paula@texcel.no)
3. Todd Bauman, U. of Maryland (bbauma1@cs.umbc.edu)
4. Anders Berglund, EBT (alb@ebt.com)
5. Karl Best, Novell (kbest@novell.com)
6. Michel Biezunski, High Text (michel@hightext.com)
7. Harvey Bingham, SGML consultant (hbingham@acm.org)
8. Steve Brown, InfoObjects (sbrown@cortland.com)
9. Martin Bryan, SGML Centre (mtbryan@sgml.u-net.com)
10. Mark Buckley, Microsoft (mbuckley@microsoft.com)
11. Len Bullard, Lockheed-Martin (cbullard@HiWAAY.net)
12. Lou Burnard, Oxford University (lou@vax.ox.ac.uk)
13. Steve Byrne, JavaSoft (steve.byrne@sun.com)
14. Kurt Conrad, Sagebrush Group (conrad@cbvcp.com)
15. Paul Cope, auto-graphics (prc@auto-graphics.com)
16. Keith Corbett, Harlequin (kmc@harlequin.com)
17. Robin Cover (robin@acadcomp.sil.org)
18. Derek Denny-Brown, TechnoTeacher (derdb@techno.com)
19. Dave Durand, Boston University (dgd@cs.bu.edu)
20. Carol Ellerbeck, INSO (Carol_Ellerbeck@inso.com)
21. Joe English, CR Laboratories (jenglish@crl.com)
22. Ralph Ferris, Fujitsu OSSI (ralph@ossi.com)
23. Lee Fogal, Digital Equipment (fogal@zk3.dec.com)
24. Todd Freter, Novell (tfreter@novell.com)
25. Matthew Fuchs, Disney Imagineering (matt@wdi.disney.com)
26. Charles Goldfarb (charles@sgmlsource.com)
27. Paul Grosso, ArborText (paul@arbortext.com)
28. Eduardo Gutentag, SunSoft (eduardo@eng.sun.com)
29. Hasse Haitto, Synex (haitto@synex.se)
30. Catherine Hamon, Hightext (hamon@hightext.com)
31. Ken Holman, Microstar (gkholman@microstar.com)
32. Rick Jelliffe, Allette Systems (ricko@allette.com.au)
33. Alan Karben, Wall Street Journal (karben@interactive.wsj.com)
34. Pär Karlsson, Ericsson (etxpkar@stdoca.ericsson.se)
35. Bill Lindsey, BDM (blindsey@BDMTech.com)
36. Per Åke Ling, Ericsson (etxperl@stdoca.ericsson.se)
37. Debbie Lapeyre, Atlis (dlapeyre@netcom.com)
38. Megan MacMillan, BDM Technologies (megan@bdmtech.com)
39. Christopher Maden, EBT (crm@ebt.com)
40. James Mason, ORNL (masonjd@ornl.gov)
41. Alex Milowski, Copernican Solutions (alex@copsol.com)
42. Steve Newcomb, TechnoTeacher (srn@techno.com)
43. Gavin Nicol, EBT (gtn@ebt.com)
44. Nancy Paisner, Hitachi (nancy@hi.com)
45. Gary Palmer, ActiveSystems (gpalmer@sonetis.com)
46. Dave Peterson, SGML Works (davep@acm.org)
47. Paul Prescod, U. of Waterloo (papresco@calum.csclub.uwaterloo.ca)

48. Lynne Price, SGML consultant (lprice@ix.netcom.com)
49. Arjun Ray, Q2-II (aray@nmds.com)
50. Bill Smith, SunSoft (bill.smith@sun.co)
51. Bob Stayton, SCO (bobs@sco.com)
52. Robert Streich, Schlumberger (streich@slb.com)
53. Jeff Suttor, SunSoft (jeff.suttor@sun.com)
54. James Tauber, U. of Western Australia (jtauber@library.uwa.edu.au)
55. Wayne Taylor, Novell (wtaylor@novell.com)
56. Henry Thompson, University of Edinburgh (ht@cogsci.ed.ac.uk)
57. Sam Wilmott, OmniMark (s.wilmott@omnimark.com)
58. Chris Wilson, Microsoft (cwilso@microsoft.com)
59. Wayne Wohler, IBM (wohler@vnet.ibm.com)
60. Lauren Wood, SoftQuad (lauren@sqwest.bc.ca)

C.2 Editorial Review Board

This specification was prepared and approved for publication by the W3C SGML Editorial Review Board (ERB). ERB approval of this specification does not necessarily imply that all ERB members voted for its approval. At the time it approved this specification, the SGML ERB had the following members:

1. Jon Bosak, Sun (jon.bosak@sun.com), chair
2. Tim Bray, Textuality (tbray@textuality.com), editor
3. James Clark (jjc@jclark.com), technical lead
4. Dan Connolly (connolly@w3.org), W3C contact
5. Steve DeRose, EBT (sjd@ebt.com)
6. Dave Hollander, HP (dmh@hpsgml.fc.hp.com)
7. Eliot Kimber, Passage Systems (kimber@passage.com)
8. Tom Magliery, NCSA (mag@ncsa.uiuc.edu)
9. Eve Maler, ArborText (elm@arbortext.com)
10. Jean Paoli, Microsoft (jeanpa@microsoft.com)
11. Peter Sharpe, SoftQuad (peter@sqwest.bc.ca)
12. C. M. Sperberg-McQueen, U. of Ill. at Chicago (cmsmcq@uic.edu), editor

IEEE Xplore RELEASE 2.1

Home | Login | Logout | Access Information

Welcome United States Patent and Trademark Office

Advanced Search

BROWSE SEARCH IEEE XPLORE GUIDE

OPTION 1
Enter keywords or phrases, select fields, and select operators

Help

sgml in All Fields

AND structured document in All Fields

AND in All Fields

» Note: If you use all three search boxes, the entries in the first two boxes take precedence over the entry in the third box.

OPTION 2
Enter keywords, phrases, or a Boolean expression

Help

» Note: You may use the search operators <and> or <or> without the start and end brackets <>.

» Learn more about Field Codes, Search Examples, and Search Operators

» Publications

Select publications

- IEEE Periodicals
- IEE Periodicals
- IEEE Conference
- IEE Conference P
- IEEE Standards

» Other Resources (Availat

- IEEE Books

» Select date range

Search latest content u

From year All

to Present

» Display Format

Citation Citatic

» Organize results

Maximum 100

Display 25 resi

Sort by Relevance

In Descending

Help Contact Us

Indexed by Inspec

Results for "((sgml<in>metadata) <and> (structured document<in>metadata))"

e-mail

Your search matched **16** of **1416205** documents.

A maximum of **100** results are displayed, **25** to a page, sorted by **Relevance** in **Descending** order.

» Search Options

View Session History

New Search

» Key

IEEE JNL	IEEE Journal or Magazine
IEE JNL	IEE Journal or Magazine
IEEE CNF	IEEE Conference Proceeding
IEE CNF	IEE Conference Proceeding
IEEE STD	IEEE Standard

Modify Search

((sgml<in>metadata) <and> (structured document<in>metadata)) Search

Check to search only within this results set

Display Format: Citation Citation & Abstract

view selected items Select All Deselect All

1. **Usability evaluation of a structured document archive**
Salminen, A.; Tiitinen, P.; Lyytikainen, V.;
System Sciences, 1999. HICSS-32. Proceedings of the 32nd Annual Hawaii In Conference on
Volume Track2, 5-8 Jan. 1999 Page(s):10 pp.
Digital Object Identifier 10.1109/HICSS.1999.772672
AbstractPlus | Full Text: PDF(212 KB) **IEEE CNF**
Rights and Permissions

2. **Structured document framework for design patterns based on SGML**
Ohtsuki, M.; Segawa, J.; Yoshida, N.; Makinouchi, A.;
Computer Software and Applications Conference, 1997. COMPSAC '97. Proce Twenty-First Annual International
13-15 Aug. 1997 Page(s):320 - 323
Digital Object Identifier 10.1109/CMPSAC.1997.624848
AbstractPlus | Full Text: PDF(508 KB) **IEEE CNF**
Rights and Permissions

3. **Standardization as a prerequisite for accessibility of electronic text inforr persons who cannot use printed material**
Bauwens, B.; Evenepoel, F.; Engelen, J.J.;
Rehabilitation Engineering, IEEE Transactions on [see also IEEE Trans. on Ne Rehabilitation]
Volume 3, Issue 1, March 1995 Page(s):84 - 89
Digital Object Identifier 10.1109/86.372897
AbstractPlus | Full Text: PDF(584 KB) **IEEE JNL**
Rights and Permissions

4. **Indexing semi-structured documents for context-based information retrie information system**
Laforest, F.; Tchounikine, A.;
Database and Expert Systems Applications, 1999. Proceedings. Tenth Interna on
1-3 Sept. 1999 Page(s):593 - 597
Digital Object Identifier 10.1109/DEXA.1999.795252
AbstractPlus | Full Text: PDF(44 KB) **IEEE CNF**
Rights and Permissions

☐ 5. **The visual specification of context**
Bruggemann-Klein, A.; Hermann, S.; Wood, D.;
Research and Technology Advances in Digital Libraries, 1999. ADL '99. Proce
Forum on
19-21 May 1999 Page(s):28 - 36
Digital Object Identifier 10.1109/ADL.1999.777688
AbstractPlus | Full Text: PDF(100 KB) **IEEE CNF**
Rights and Permissions

☐ 6. **Automatic generation algorithm of uniform DTD for structured document:**
Chun-Sik Yoo; Seon-Mi Woo; Yong-Sung Kim;
TENCON 99. Proceedings of the IEEE Region 10 Conference
Volume 2, 15-17 Sept. 1999 Page(s):1095 - 1098 vol.2
Digital Object Identifier 10.1109/TENCON.1999.818614
AbstractPlus | Full Text: PDF(240 KB) **IEEE CNF**
Rights and Permissions

☐ 7. **Index design for structured documents based on abstraction**
Chow, J.-H.; Cheng, J.; Chang, D.; Xu, J.;
Database Systems for Advanced Applications, 1999. Proceedings., 6th Interna
on
19-21 April 1999 Page(s):89 - 96
Digital Object Identifier 10.1109/DASFAA.1999.765740
AbstractPlus | Full Text: PDF(168 KB) **IEEE CNF**
Rights and Permissions

☐ 8. **An indexing model for structured documents to support queries on conte and attributes**
Tuong Dao;
Research and Technology Advances in Digital Libraries, 1998. ADL 98. Procee
International Forum on
22-24 April 1998 Page(s):88 - 97
Digital Object Identifier 10.1109/ADL.1998.670383
AbstractPlus | Full Text: PDF(224 KB) **IEEE CNF**
Rights and Permissions

☐ 9. **Hypermedia document and workflow management based on active object databases**
Kappel, G.; Rausch-Schott, S.; Reich, S.; Retschitzegger, W.;
System Sciences, 1997, Proceedings of the Thirtieth Hawaii International Conf
Volume 4, 7-10 Jan. 1997 Page(s):377 - 386 vol.4
Digital Object Identifier 10.1109/HICSS.1997.663410
AbstractPlus | Full Text: PDF(1296 KB) **IEEE CNF**
Rights and Permissions

☐ 10. **A graphical environment for change detection in structured documents**
Chang, G.J.S.; Patel, G.; Relihan, L.; Wang, J.T.L.;
Computer Software and Applications Conference, 1997. COMPSAC '97. Proce
Twenty-First Annual International
13-15 Aug. 1997 Page(s):536 - 541
Digital Object Identifier 10.1109/CMPSAC.1997.625064
AbstractPlus | Full Text: PDF(480 KB) **IEEE CNF**
Rights and Permissions

☐ 11. **OOHS: an object-oriented hypermedia system**
HyunKi Kim; HakGene Shin; JaeWoo Chang;
Computer Software and Applications Conference, 1996. COMPSAC '96., Proc
International
21-23 Aug. 1996 Page(s):496 - 501
Digital Object Identifier 10.1109/CMPSAC.1996.544619

AbstractPlus | Full Text: PDF(604 KB) **IEEE CNF**
Rights and Permissions

12. **An efficient algorithm to compute differences between structured docum**
Kyong-Ho Lee; Yoon-Chul Choy; Sung-Bae Cho;
Knowledge and Data Engineering, IEEE Transactions on
Volume 16, Issue 8, Aug. 2004 Page(s):965 - 979
Digital Object Identifier 10.1109/TKDE.2004.19
AbstractPlus | References | Full Text: PDF(1304 KB) **IEEE JNL**
Rights and Permissions

13. **Design and implementation of a structured information retrieval system f documents**
Sung-Geun Han; Jeong-Han Son; Jae-Woo Chang; Zong-Cheol Zhoo;
Database Systems for Advanced Applications, 1999. Proceedings., 6th Interna Conference on
19-21 April 1999 Page(s):81 - 88
Digital Object Identifier 10.1109/DASFAA.1999.765739
AbstractPlus | Full Text: PDF(164 KB) **IEEE CNF**
Rights and Permissions

14. **Semantic structuring of documents**
Poullet, L.; Pinon, J.-M.; Calabretto, S.;
Information Technology, 1997. BIWIT '97., Proceedings of the Third Basque In Workshop on
2-4 July 1997 Page(s):118 - 124
Digital Object Identifier 10.1109/BIWIT.1997.614058
AbstractPlus | Full Text: PDF(508 KB) **IEEE CNF**
Rights and Permissions

15. **A heuristics-based approach to query optimization in structured docume**
Che, D.; Aberer, K.;
Database Engineering and Applications, 1999. IDEAS '99. International Sympo Proceedings
2-4 Aug. 1999 Page(s):24 - 33
Digital Object Identifier 10.1109/IDEAS.1999.787248
AbstractPlus | Full Text: PDF(164 KB) **IEEE CNF**
Rights and Permissions

16. **Query by templates: a generalized approach for visual query formulation dominated databases**
Sengupta, A.; Dillon, A.;
Research and Technology Advances in Digital Libraries, 1997. ADL '97. Proce International Forum on
7-9 May 1997 Page(s):36 - 47
Digital Object Identifier 10.1109/ADL.1997.601198
AbstractPlus | Full Text: PDF(1580 KB) **IEEE CNF**
Rights and Permissions

Help Contact Us Privacy & :

Indexed by Inspec

Results for **"(sgml<in>metadata)"**
Your search matched **94** of **1416205** documents.
A maximum of **100** results are displayed, **25** to a page, sorted by **Relevance** in **Descending** order.

» Search Options

View Session History

New Search

» Other Resources
(Available For Purchase)

Top Book Results

Handbook for Preparing Engineering Documents
by Nagle, J. G.;
Paperback, Edition: 1

View All 1 Result(s)

» Key

IEEE JNL	IEEE Journal or Magazine
IEE JNL	IEE Journal or Magazine
IEEE CNF	IEEE Conference Proceeding
IEE CNF	IEE Conference Proceeding
IEEE STD	IEEE Standard

Modify Search

(sgml<in>metadata) Search

☐ Check to search only within this results set

Display Format: ◉ Citation ○ Citation & Abstract

view selected items **Select All** **Deselect All** **View: 1-25** | 26-

☐ 1. **A standard-based approach to text and hypertext mutual conversion and**
Zheng, M.; Rada, R.;
Professional Communication Conference, 1993. IPCC 93 Proceedings. 'The N Technical Communication: People, Processes, Products'
5-8 Oct. 1993 Page(s):116 - 121
Digital Object Identifier 10.1109/IPCC.1993.593791
AbstractPlus | Full Text: PDF(416 KB) **IEEE CNF**
Rights and Permissions

☐ 2. **Succession in standardization: Grafting XML onto SGML**
Egyedi, T.M.; Loeffen, A.G.A.J.;
Standardization and Innovation in Information Technology, 2001 2nd IEEE Coi
3-6 Oct. 2001 Page(s):38 - 49
Digital Object Identifier 10.1109/SIIT.2001.968553
AbstractPlus | Full Text: PDF(882 KB) **IEEE CNF**
Rights and Permissions

☐ 3. **Object databases for SGML document management**
Olson, M.R.; Lee, B.S.;
System Sciences, 1997, Proceedings of the Thirtieth Hawaii International Conf
Volume 3, 7-10 Jan. 1997 Page(s):39 - 48 vol.3
Digital Object Identifier 10.1109/HICSS.1997.661568
AbstractPlus | Full Text: PDF(1216 KB) **IEEE CNF**
Rights and Permissions

☐ 4. **ODIL: an SGML description language of the layout structure of document**
Lefevre, P.; Reynaud, F.;
Document Analysis and Recognition, 1995., Proceedings of the Third Internati on
Volume 1, 14-16 Aug. 1995 Page(s):480 - 488 vol.1
Digital Object Identifier 10.1109/ICDAR.1995.599040
AbstractPlus | Full Text: PDF(896 KB) **IEEE CNF**
Rights and Permissions

☐ 5. **Java/CORBA groupware for SGML documents**
Geyer, J.; Chopping, E.; Bossomaier, T.;
Software Engineering: Education and Practice, 1998. Proceedings. 1998 Interr Conference

26-29 Jan. 1998 Page(s):166 - 171
Digital Object Identifier 10.1109/SEEP.1998.707647

AbstractPlus | Full Text: PDF(40 KB) **IEEE CNF**
Rights and Permissions

☐ 6. **SGML nets: integrating document and workflow modeling**
Weitz, W.;
System Sciences, 1998., Proceedings of the Thirty-First Hawaii International C
Volume 2, 6-9 Jan. 1998 Page(s):185 - 194 vol.2
Digital Object Identifier 10.1109/HICSS.1998.651699

AbstractPlus | Full Text: PDF(164 KB) **IEEE CNF**
Rights and Permissions

☐ 7. **Automatic generation algorithm of uniform DTD for structured document**
Chun-Sik Yoo; Seon-Mi Woo; Yong-Sung Kim;
TENCON 99. Proceedings of the IEEE Region 10 Conference
Volume 2, 15-17 Sept. 1999 Page(s):1095 - 1098 vol.2
Digital Object Identifier 10.1109/TENCON.1999.818614

AbstractPlus | Full Text: PDF(240 KB) **IEEE CNF**
Rights and Permissions

☐ 8. **SGML for e-governance: the case of the Finnish Parliament**
Salminen, A.; Lyytikainen, V.; Tiitinen, P.; Mustajarvi, O.;
Database and Expert Systems Applications, 2000. Proceedings. 11th Internati
4-8 Sept. 2000 Page(s):349 - 353
Digital Object Identifier 10.1109/DEXA.2000.875050

AbstractPlus | Full Text: PDF(388 KB) **IEEE CNF**
Rights and Permissions

☐ 9. **Multimedia interchange using SGML/HyTime. I. Structures**
Newcomb, S.R.;
Multimedia, IEEE
Volume 2, Issue 2, Summer 1995 Page(s):86 - 89
Digital Object Identifier 10.1109/93.388212

AbstractPlus | References | Full Text: PDF(348 KB) **IEEE JNL**
Rights and Permissions

☐ 10. **Using SGML to create complex interactive documents for electronic publ**
Goldie, P.;
Professional Communication, IEEE Transactions on
Volume 40, Issue 2, June 1997 Page(s):129 - 138
Digital Object Identifier 10.1109/47.588825

AbstractPlus | References | Full Text: PDF(244 KB) **IEEE JNL**
Rights and Permissions

☐ 11. **Experiences of SGML standardization: the case of the Finnish legislative**
Salminen, A.; Lyytikainen, V.; Tiitinen, P.; Mustajarvi, O.;
System Sciences, 2001. Proceedings of the 34th Annual Hawaii International (
Jan 3-6 2001 Page(s):9 pp.

AbstractPlus | Full Text: PDF(216 KB) **IEEE CNF**
Rights and Permissions

☐ 12. **Design and implementation of a structured information retrieval system f documents**
Sung-Geun Han; Jeong-Han Son; Jae-Woo Chang; Zong-Cheol Zhoo;
Database Systems for Advanced Applications, 1999. Proceedings., 6th Interna Conference on
19-21 April 1999 Page(s):81 - 88
Digital Object Identifier 10.1109/DASFAA.1999.765739

AbstractPlus | Full Text: PDF(164 KB) **IEEE CNF**

☐ 13. **Structured document framework for design patterns based on SGML**
Ohtsuki, M.; Segawa, J.; Yoshida, N.; Makinouchi, A.;
Computer Software and Applications Conference, 1997. COMPSAC '97. Proce
Twenty-First Annual International
13-15 Aug. 1997 Page(s):320 - 323
Digital Object Identifier 10.1109/CMPSAC.1997.624848

AbstractPlus | Full Text: PDF(508 KB) **IEEE CNF**

☐ 14. **Conceptual levels of SGML tags: a proposed taxonomy based on the tag**
Orlando Project
Ruecker, S.;
Web Information Systems Engineering, 2000. Proceedings of the First Internat
on
Volume 2, 19-21 June 2000 Page(s):2 - 10 vol.2
Digital Object Identifier 10.1109/WISE.2000.882843

AbstractPlus | Full Text: PDF(788 KB) **IEEE CNF**

☐ 15. **Conversion of documents to and from SGML**
Hunter, B.;
Adding Value to Documents with Markup Languages, IEE Colloquium on
1994 Page(s):5/1 - 5/4

AbstractPlus | Full Text: PDF(236 KB) **IEE CNF**

☐ 16. **An SGML based viewer for form documents**
Atalay, V.; Arslan, E.;
Document Analysis and Recognition, 1999. ICDAR '99. Proceedings of the Fift
Conference on
20-22 Sept. 1999 Page(s):201 - 204
Digital Object Identifier 10.1109/ICDAR.1999.791759

AbstractPlus | Full Text: PDF(52 KB) **IEEE CNF**

☐ 17. **Legal citation referencing using SGML and HyTime**
Ebenhoch, P.;
Database and Expert Systems Applications, 1998. Proceedings. Ninth Internat
on
26-28 Aug. 1998 Page(s):625 - 630
Digital Object Identifier 10.1109/DEXA.1998.707468

AbstractPlus | Full Text: PDF(44 KB) **IEEE CNF**

☐ 18. **Error tolerant document structure analysis**
Klein, B.; Fankhauser, P.;
Research and Technology Advances in Digital Libraries, 1997. ADL '97. Proce
International Forum on
7-9 May 1997 Page(s):116 - 127
Digital Object Identifier 10.1109/ADL.1997.601207

AbstractPlus | Full Text: PDF(972 KB) **IEEE CNF**

☐ 19. **An object-based methodology for knowledge representation in SGML**
Kelsey, R.L.; Hartley, R.T.; Webster, R.B.;
Tools with Artificial Intelligence, 1997. Proceedings., Ninth IEEE International (
3-8 Nov. 1997 Page(s):304 - 311

Digital Object Identifier 10.1109/TAI.1997.632270

AbstractPlus | Full Text: PDF(632 KB) **IEEE CNF**
Rights and Permissions

☐ 20. **Toward automated document reformatting: a SGML markup system**
Lee, J.C.; Swietlik, C.E.;
System Theory, 1996., Proceedings of the Twenty-Eighth Southeastern Symp
31 March-2 April 1996 Page(s):137 - 141
Digital Object Identifier 10.1109/SSST.1996.493486

AbstractPlus | Full Text: PDF(388 KB) **IEEE CNF**
Rights and Permissions

☐ 21. **IEE Colloquium on ` Adding Value to Documents with Markup Language no.1994/142)**
Adding Value to Documents with Markup Languages, IEE Colloquium on
1994

AbstractPlus | Full Text: PDF(20 KB) **IEE CNF**

☐ 22. **Optimal and suboptimal broad-band source location estimation**
Schultheiss, P.M.; Messer, H.;
Signal Processing, IEEE Transactions on [see also Acoustics, Speech, and Si IEEE Transactions on]
Volume 41, Issue 9, Sept. 1993 Page(s):2752 - 2763
Digital Object Identifier 10.1109/78.236500

AbstractPlus | Full Text: PDF(844 KB) **IEEE JNL**
Rights and Permissions

☐ 23. **Indexing semi-structured documents for context-based information retri information system**
Laforest, F.; Tchounikine, A.;
Database and Expert Systems Applications, 1999. Proceedings. Tenth Interna on
1-3 Sept. 1999 Page(s):593 - 597
Digital Object Identifier 10.1109/DEXA.1999.795252

AbstractPlus | Full Text: PDF(44 KB) **IEEE CNF**
Rights and Permissions

☐ 24. **Of document databases, SGML, and rhetorical neutrality**
Weiss, E.H.;
Professional Communication, IEEE Transactions on
Volume 36, Issue 2, June 1993 Page(s):58 - 61
Digital Object Identifier 10.1109/47.222682

AbstractPlus | Full Text: PDF(440 KB) **IEEE JNL**
Rights and Permissions

☐ 25. **Lessons learned prototyping an SGML-based computerized document m system**
Madigan, C.; Silber, M.K.; Wilson, S.;
Professional Communication, IEEE Transactions on
Volume 40, Issue 2, June 1997 Page(s):139 - 143
Digital Object Identifier 10.1109/47.588831

AbstractPlus | References | Full Text: PDF(36 KB) **IEEE JNL**
Rights and Permissions

View: 1-25 | 26-

Help Contact Us Privacy &

Of Document Databases, SGML, and Rhetorical Neutrality

Edmond H. Weiss

***Abstract*—Traditionally, rhetorical skill resides in the speaker/sender. Rhetorical effectiveness—cogency, clarity, persuasiveness—is the product of the speaker's resourcefulness in adapting to the setting and audience. In the current epoch, though, new technology has enabled the *receiver to shape the message.* Today, publishing technical information often consists in letting the receivers search the files, extract what *they* judge relevant, sequence and organize it any way they wish, and even print or display it to their own specifications. Often, today's writer is NOT creating deliberately worded and presented messages but, rather, feeding molecular articles to "rhetorically neutral" databases, from which readers may extract what they wish. Such technologies as SGML even further limit writers, depriving them of such basic presentation devices as deciding where pages will begin and end. What are the rhetorical implications of technology that "empowers" readers and "enfeebles" writers?**

INTRODUCTION

Speech is linear and temporal. The speaker or "rhetor" invents a sequence of words and utters them through a unique series of moments. The listener has only two choices: to hear what is said—with the sequence and pacing the rhetor chooses—or not to hear. This point is logically prior to a more exotic discussion of the vagaries of meaning and interpretation; at issue here is not what the listener understands or believes. What matters first is what is heard: particular words, in unique sequence, with deliberate cadences and pauses, and even resourcefully selected suprasegmentals—stress, pitch, and intonation.

If a speech fails, does not persuade or win assent, we may fault the rhetor for lack of resourcefulness or for ineffectiveness. Unless we conclude that most listeners did not hear the message, we cannot ascribe the failure of the communication to the actions of the listeners.

The foundation of rhetorical craft, the context of rhetorical criticism, is that rhetorical power resides mainly, if not entirely, in the speaker. The peculiar linear-temporal nature of speaking and listening means that the rhetor operates the transaction of meaning. In effect, the sender of the message controls the production and distribution and reception of the message. So, when we teach speaking, we presume that the speaker alone can implement our advice and methods.

WRITING AS AN APPROXIMATION OF SPEECH

When we extend the domain of rhetoric to writing—not just to persuasion, but to all kinds of expository and instructional writing—we adapt notions of speaking to notions of writing. This adaptation is sometimes direct and literal; the rules of correct grammar are quite similar, for example. But more often, the adaptation is by analogy.

When we call writing "written speech," we are not speaking literally at all. Reading a document of more than a few words is not merely different from listening; it is almost entirely different. As has been discussed frequently [1], readers are active collaborators in the writing-reading transaction, much more so than listeners to speech. Readers have styles and methods; they scan, skim, construct, and deconstruct. Even in relatively short messages, they alter the order of the text by starting in the middle (or at the end) and arrogate to themselves the right to choose the sequence of the paragraphs, pages, and chapters.

This proclivity in the reader, this active intervention in the pacing and arrangement of the message, is a central issue in the application of rhetorical craft to technical and business communication. On the one hand, many of us have dedicated our teaching and consulting careers to helping writers wage war against the readers' proclivities by removing from documents any writing or publication practices that would encourage them to skip, branch, loop, or otherwise deconstruct the message. [2] That is, given the inclination of the readers to invent their own versions of the message, and given the inevitability of their doing so, we have at least tried to alter those writing and presentation practices that would oblige them to construct their own message from disjointed elements within our text.

At the first level, the sentence, we argue for particular verb forms and sentence patterns that invite the sentence to be read as written, rather than rebuilt. (For example, writing conditional instructions in the If-Then form, rather than the too-common Then-If, reduces the need to re-read such sentences.) [3] Some of us have also lobbied strenuously for periodic sentences [4] and paragraphs, knowing that a series of loose sentences encourages skimming. Similarly, we have counseled simplicity and "readability," knowing that dense, impenetrable prose encourages readers to compile their own distilled versions of our messages.

At higher levels of organization, we have also warned against disjointedness in publications, in particular the practices of referring people to other chapters or documents and the insidious practice of discussing figures, charts, and tables that are not adjacent to the text that cites them [5].

All this counsel, all these techniques and principles, are to some extent an attempt to simulate in writing and page

Manuscript received November 1992.
The author is at 1612 Crown Point Lane, Cherry Hill, NJ 08003.
IEEE Log Number 9209277.

0361-1434/93$03.00

design some of the effects that work naturally in real speech, where, for example, the receivers simply cannot jump ahead or ignore the sequence of the statements. It is not too much of a simplification to say that most principles of style reflect an attempt to give the writer some of the manipulative and controlling power of the speaker.

Much of this effort to extend the rhetor's control to the domain of writing is manifest in the technology nowadays called "publishing": the design of pages that encourages the reader to look at the right information in the right sequence. Throughout the 1980s, the tools used for word processing have become increasingly publication-intensive. Today's full-featured word processors will scarcely work at all without a "printer driver," a table of instructions that converts the data in the file to particular page output. From the 70s onward, the trend has been unmistakable: originators have become publishers. Today, a professional who scribbles reports on yellow tablets and lets someone else render the finished document is not only rare but somewhat perverse. And the professional technical writer of today can shape the precise look and feel of a page with more freedom and flexibility than any traditional typesetter could have.

Again, the rationale underlying publishing, the rules and heuristics that prefer one column width and typeface over another [6], are adaptations by analogy of the rhetor's methods for controlling the pace and sequence of the presentation, for controlling what will be attended to, and in what order. The widespread popularity of the 4 1/2" column of text, for example, reflects the knowledge that wider columns encourage skimming. The deliberate use of text highlights and emphases is the typographic equivalent of the speaker's loudness, silence, and other devices of emphasis. Everyday design choices such as the amount of white space, the typeface and font, the position of page-breaks . . . all are meant to produce a more usable document, which, in many ways, is a euphemism for a document that will be read as the *writer intended.*

An Alternative Rhetoric

Please note that the last few paragraphs have been "on the one hand." Some of us, as I said, have done what we could to inhibit the reader's independence, to control paternalistically the pattern of attention, and thereby enhance the reliability of the transaction. On the other hand, though, are many who find it inappropriate and unenlightened to write with a traditional rhetor's expectations of audience control and manipulation. Many current students of rhetoric are far more concerned with the multidimensional nature of reading (the kind we associate with hypertext, for example). They are much less interested in notions of writing and publishing that are meant to constrain or compel the reader than in "rhetorically neutral" forms of written messages that can be extracted, manipulated, and reconfigured according to the needs, curiosity, and resourcefulness of the reader.

This alternative rhetoric changes the writer-reader dynamic. The skillful writer is the creator of a database of words and pictures, organized so that readers can find and read what they want and can link pages and sections in ways the author could not have imagined. All documents are "virtual," existing potentially in the database but not in any ordinary physical reality.

The most familiar model for this alternative, rhetorically unconstrained form of reading is literary or biblical scholarship, in which one phrase suggests another to a reader so familiar with a library of scholarly materials that he or she can jump from volume to volume and create a web of associations. (Traditionally, there is even a certain cachet to knowing much of this library by heart: "Real Talmudists don't use CD-ROMs.")

This alternative rhetoric is exhilarating, and even somewhat visionary. It anticipates a populist era in which anyone with the right tools can tell you how many times (and where) the Bible mentions, say, "wine," or how many of Shakespeare's plots involve sending a letter, or which telephone numbers are unassigned in a given area code, or how to save a run-time version of a presentation, or what the odds are for this week's football games. They even expect that readers may pursue all these questions and receive all these answers in the same reading "session," while never leaving their bookless desks.

In this new communication culture, readers have nearly endless freedom in modifying their reading displays and in printing their personally constructed version of what they have read. With my own hypertext version of the Bible for example, I can set the typeface, the font size, and the column width; I can look at one version at a time, or shrink the panels to allow two or more versions at the same time—including Hebrew for the Old Testament and Greek for the New. I can export any subset of verses into my word processor, and if I think some of the commandments need changing, I can *edit them*!

Like everyone else who works with Microsoft Windows 3.1, or comparable systems, I have extraordinary powers of searching, sharing, and linking data. But I also have the power to produce unreadable, badly-designed displays and pages that would make the talented writers at Microsoft and Microsoft Press cringe! If, for example, I "maximize" my "Help Window," I can generate rows of text with 120 characters. (If I tweaked my setup a bit, I probably could get 200.) Or, alternately, I can produce a column so narrow, or a window so small, that almost nothing can be read without scrolling.

In the 90s, then, the author creates a library of molecules and modules of text and picture (not manuals and reports), and the readers—aided by linking, searching, and display tools—manufacture their own versions of the message. The sender remains rhetorically neutral; the readers persuade and instruct themselves.

This, say the enthusiasts for the alternative rhetoric, is really what readers and audiences have always done anyway. Learning and persuasion only occur when receivers *instruct and persuade themselves*, we often hear. No one reads the sentences in a technical publication in order. Now, though, the communicators and senders embrace the idea, instead of imposing irrelevant notions of speech. Now, writers and readers collaborate to do away with ordinary publications, which, they argue, tend to inhibit and frustrate all but the most scholarly readers' treks through hyperspace.

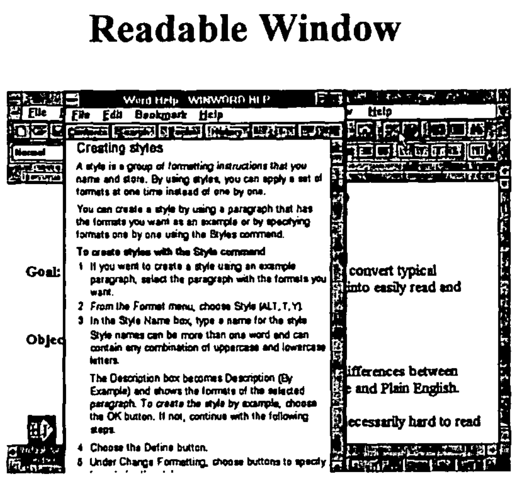


Fig. 1. Readable window.

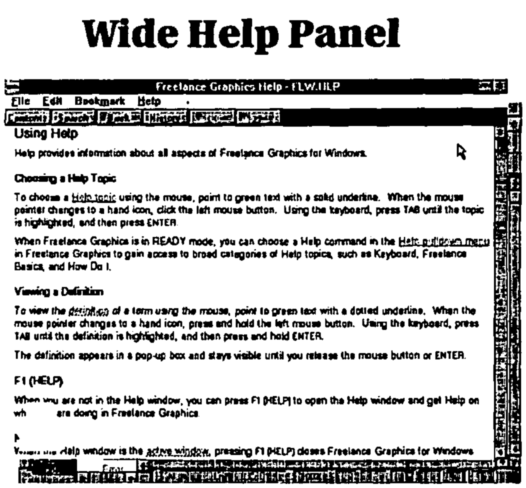


Fig. 2. Wide window.

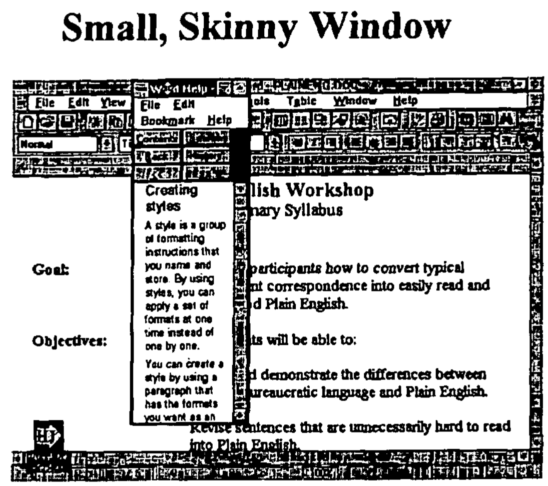


Fig. 3. Small, skinny window.

SGML

Among the most interesting things about this new rhetorical culture, which often has a futuristic or even "New Age" cast to it, is that it has found an important ally in, of all places, the military. For quite different reasons, the Department of Defense (DoD) is also generally opposed to publications in which the document is linked to a particular piece of software and printer driver. Put simply, the military wants all future contractors to provide documents that are "compliant" with its CALS program (Computer-Aided Acquisition and Logistic Support) [7]. Instead of paper publications, instead of WordPerfect files, the military wants *SGML* (Standard Generalized Markup Language) documents. It is not only a broad approach to publication management, but also a specific ISO standard, and a Department of Defense standard as well [8]. SGML breaks the connection between a document and the software and hardware associated with the creation of that document. It reduces the document, no matter how it was created, to ASCII text and requires that all formatting information be rendered as ASCII tags, typically inserted between the < > brackets.

By severing the link between the document and its software and hardware, SGML lets publications be output to any printer . . . or screen, or speech synthesizer, or automatic translator, or archive storage device. Its codes and tags identify the logical and structural components in a document. But, unlike the codes familiar in word processing systems, SGML tags are NOT supposed to refer to specific typefaces, fonts, sizes, margins, tabs, leading, text highlights, page sizes, etc.

Instead of inserting <TimesRoman 24><center><bold> before the main title of an article, the SGML author inserts <maintitle>. Elsewhere, an output device consults a dictionary of Document Type Definitions (DTD's) and decides what combination of screen or printer instructions should be attached to "maintitle." These DTD's, written generally by someone other than the author, include formal definitions of document components and elements (using data dictionary logic) and hierarchical rules for what components may follow others.

The Dictionary of DTD's contains not only the current set of rules for the typefaces and indentation of headings; it also contains logical rules that prevent an author from skipping a level of indentation (for example, by going directly from <2ndhead> to <4thhead> without the intervening step). It also establishes scores of other precedence rules, such as forbidding a footnote in a caption or superscript in a footnote—if that's what the DTD developers wish.

So, curiously, the practical, administrative goals of the SGML/CALS enthusiasts dovetail nicely with the humanistic and liberal goals of the hypertext rhetoricians. Eventually, for example, the Navy wants to be able to download from satellites recent technical information from the files of its contractors to the workstations on its ships—instead of needing to carry 5 tons of manuals on a 90-ton vessel. How remarkably similar to what Ted Nelson wants for his Xanadu network! [9]

IMPLICATIONS FOR THE PROFESSIONAL COMMUNICATOR

There is a kind of fateful inevitability and technological elegance to the hypertext-document database-SGML trend. There is no point in resisting or opposing any trend attractive to both the military and the mystical! Indeed, the new technologies will probably create thousands of new, highly skilled jobs for technical communicators.

But during this transitional time, as the old rhetoric of sender-controlled writing yields to the new, we might ask ourselves a few technical and philosophical questions.

Does the writer/publisher of a document have an obligation to reduce the reading burden for ingenuous readers? That is, when we give readers the option to display and print our text in any way they like, should we be content with their sometimes unwise choices? Is it condescending to observe that many users of computers, especially those who use multi-panel screens, make poor choices: absurd and stressful colors; hard-to-read typefaces (when more readable ones are near at hand); counterproductive column widths, and so forth. If many of our readers are ingenuous about readability and page design, should we not protect them from themselves? If we know that badly designed pages and displays lead to fatigue, errors, and waste, can we encourage just such pages? Must we, like good citizens in a democracy, accept the right of readers to be wrong?

Can large publishing institutions be trusted to choose enlightened publishing standards? If we turn over our documents to our clients with only SGML tags, can we trust them to choose appropriate DTD's? Is there any historical evidence, for instance, that the federal government's principal publishers—like DoD or NTIS—appreciate the importance of typography and page design? Do we, as authors, have no right to demand that our articles and books look clear, accessible, and professional?

Can writing be an intellectually, professionally rewarding occupation if we break the traditional rhetorical link between the writer and reader? What professional writer would choose a career of putting thousands of chunks of information into a database stew? What pleasure is left in the process without a sense of the audience, the feeling that I am "speaking" words of my choice, in an order of my choice, to a person I recognize?

Technical communication is not a literary enterprise. In most cases, manuals and other publications must be produced by teams, with the individual contributors sacrificing their writerly egos to the overall project. Moreover, technical publications are hardly ever complete, and like computer programs, they are often modified and "maintained" by other writers. Indeed, it appears that manuals written entirely by one person—a method rarely used in sophisticated publication organizations—may be problematical because they are inherently unmaintainable. Even under these conditions, though, most professional writers still want to affect their readers in predictable ways, to practice their hard-learned craft, and to point to some products or creations that are more actual than virtual. Absent these small satisfactions, it is hard to imagine anyone wanting to be a technical writer at all.

REFERENCES

[1] M. Coney, "Technical readers and their rhetorical roles," in *IEEE Trans. Prof. Commun.*, vol. 35, pp. 58-63, 1992.
[2] E. Weiss, "Usability: Toward a science of user documentation," in *Computerworld, In Depth 9-16*, Jan. 1983.
[3] E. Weiss, "Ten Ways to Write an Unclear Instruction," in *Data Training*, June 1983.
[4] W. Strunk and E. White, *The Elements of Style 3/e.* New York: Macmillan, 1979, ch. 2.
[5] E. Weiss, *How to Write Usable User Documentation 2/e.* Phoenix: Oryx Press, 1991.
[6] D. Felker *et al.*, "Typographic Principles," in *Guidelines for Document Designers.* Washington, DC: American Institutes for Research, 1981.
[7] *Military Handbook, Department of Defense Computer-Aided Acquisition and Logistic Support (CALS) Program Implementation Guide*, MIL-HDBK, Dec. 1988.
[8] SGML is defined in ISO 8879 and Department of Defense MIL-M-2801.
[9] Ted Nelson, *Literary Machines*, Ed. 87.1, Self-published, 1987

Edmond H. Weiss, Ph.D. is an independent writer, lecturer, and consultant who travels North America teaching seminars on business and technical communication. He is the author of *One Hundred Writing Remedies: Practical Exercises for Technical Writing* (Oryx Press 1990), *How to Write Usable User Documentation* (2nd edition, Oryx Press 1991), and *The ISO 9000 Quality Manual: A Handbook/Workbook* (Edmond Weiss Seminars 1993). His base is Cherry Hill, NJ.

Of Document Databases, SGML, and Rhetorical Neutrality

Edmond H. Weiss

***Abstract*—Traditionally, rhetorical skill resides in the speaker/sender. Rhetorical effectiveness—cogency, clarity, persuasiveness—is the product of the speaker's resourcefulness in adapting to the setting and audience. In the current epoch, though, new technology has enabled the *receiver to shape the message.* Today, publishing technical information often consists in letting the receivers search the files, extract what *they* judge relevant, sequence and organize it any way they wish, and even print or display it to their own specifications. Often, today's writer is NOT creating deliberately worded and presented messages but, rather, feeding molecular articles to "rhetorically neutral" databases, from which readers may extract what they wish. Such technologies as SGML even further limit writers, depriving them of such basic presentation devices as deciding where pages will begin and end. What are the rhetorical implications of technology that "empowers" readers and "enfeebles" writers?**

Introduction

Speech is linear and temporal. The speaker or "rhetor" invents a sequence of words and utters them through a unique series of moments. The listener has only two choices: to hear what is said—with the sequence and pacing the rhetor chooses—or not to hear. This point is logically prior to a more exotic discussion of the vagaries of meaning and interpretation; at issue here is not what the listener understands or believes. What matters first is what is heard: particular words, in unique sequence, with deliberate cadences and pauses, and even resourcefully selected suprasegmentals—stress, pitch, and intonation.

If a speech fails, does not persuade or win assent, we may fault the rhetor for lack of resourcefulness or for ineffectiveness. Unless we conclude that most listeners did not hear the message, we cannot ascribe the failure of the communication to the actions of the listeners.

The foundation of rhetorical craft, the context of rhetorical criticism, is that rhetorical power resides mainly, if not entirely, in the speaker. The peculiar linear-temporal nature of speaking and listening means that the rhetor operates the transaction of meaning. In effect, the sender of the message controls the production and distribution and reception of the message. So, when we teach speaking, we presume that the speaker alone can implement our advice and methods.

Writing as an Approximation of Speech

When we extend the domain of rhetoric to writing—not just to persuasion, but to all kinds of expository and instructional writing—we adapt notions of speaking to notions of writing. This adaptation is sometimes direct and literal; the rules of correct grammar are quite similar, for example. But more often, the adaptation is by analogy.

When we call writing "written speech," we are not speaking literally at all. Reading a document of more than a few words is not merely different from listening; it is almost entirely different. As has been discussed frequently [1], readers are active collaborators in the writing-reading transaction, much more so than listeners to speech. Readers have styles and methods; they scan, skim, construct, and deconstruct. Even in relatively short messages, they alter the order of the text by starting in the middle (or at the end) and arrogate to themselves the right to choose the sequence of the paragraphs, pages, and chapters.

This proclivity in the reader, this active intervention in the pacing and arrangement of the message, is a central issue in the application of rhetorical craft to technical and business communication. On the one hand, many of us have dedicated our teaching and consulting careers to helping writers wage war against the readers' proclivities by removing from documents any writing or publication practices that would encourage them to skip, branch, loop, or otherwise deconstruct the message. [2] That is, given the inclination of the readers to invent their own versions of the message, and given the inevitability of their doing so, we have at least tried to alter those writing and presentation practices that would oblige them to construct their own message from disjointed elements within our text.

At the first level, the sentence, we argue for particular verb forms and sentence patterns that invite the sentence to be read as written, rather than rebuilt. (For example, writing conditional instructions in the If-Then form, rather than the too-common Then-If, reduces the need to re-read such sentences.) [3] Some of us have also lobbied strenuously for periodic sentences [4] and paragraphs, knowing that a series of loose sentences encourages skimming. Similarly, we have counseled simplicity and "readability," knowing that dense, impenetrable prose encourages readers to compile their own distilled versions of our messages.

At higher levels of organization, we have also warned against disjointedness in publications, in particular the practices of referring people to other chapters or documents and the insidious practice of discussing figures, charts, and tables that are not adjacent to the text that cites them [5].

All this counsel, all these techniques and principles, are to some extent an attempt to simulate in writing and page

Manuscript received November 1992.
The author is at 1612 Crown Point Lane, Cherry Hill, NJ 08003.
IEEE Log Number 9209277.

0361-1434/93$03.00 © 1993 IEEE

design some of the effects that work naturally in real speech, where, for example, the receivers simply cannot jump ahead or ignore the sequence of the statements. It is not too much of a simplification to say that most principles of style reflect an attempt to give the writer some of the manipulative and controlling power of the speaker.

Much of this effort to extend the rhetor's control to the domain of writing is manifest in the technology nowadays called "publishing": the design of pages that encourages the reader to look at the right information in the right sequence. Throughout the 1980s, the tools used for word processing have become increasingly publication-intensive. Today's full-featured word processors will scarcely work at all without a "printer driver," a table of instructions that converts the data in the file to particular page output. From the 70s onward, the trend has been unmistakable: originators have become publishers. Today, a professional who scribbles reports on yellow tablets and lets someone else render the finished document is not only rare but somewhat perverse. And the professional technical writer of today can shape the precise look and feel of a page with more freedom and flexibility than any traditional typesetter could have.

Again, the rationale underlying publishing, the rules and heuristics that prefer one column width and typeface over another [6], are adaptations by analogy of the rhetor's methods for controlling the pace and sequence of the presentation, for controlling what will be attended to, and in what order. The widespread popularity of the 4 1/2" column of text, for example, reflects the knowledge that wider columns encourage skimming. The deliberate use of text highlights and emphases is the typographic equivalent of the speaker's loudness, silence, and other devices of emphasis. Everyday design choices such as the amount of white space, the typeface and font, the position of page-breaks . . . all are meant to produce a more usable document, which, in many ways, is a euphemism for a document that will be read as the *writer intended.*

An Alternative Rhetoric

Please note that the last few paragraphs have been "on the one hand." Some of us, as I said, have done what we could to inhibit the reader's independence, to control paternalistically the pattern of attention, and thereby enhance the reliability of the transaction. On the other hand, though, are many who find it inappropriate and unenlightened to write with a traditional rhetor's expectations of audience control and manipulation. Many current students of rhetoric are far more concerned with the multidimensional nature of reading (the kind we associate with hypertext, for example). They are much less interested in notions of writing and publishing that are meant to constrain or compel the reader than in "rhetorically neutral" forms of written messages that can be extracted, manipulated, and reconfigured according to the needs, curiosity, and resourcefulness of the reader.

This alternative rhetoric changes the writer-reader dynamic. The skillful writer is the creator of a database of words and pictures, organized so that readers can find and read what they want and can link pages and sections in ways the author could not have imagined. All documents are "virtual," existing potentially in the database but not in any ordinary physical reality.

The most familiar model for this alternative, rhetorically unconstrained form of reading is literary or biblical scholarship, in which one phrase suggests another to a reader so familiar with a library of scholarly materials that he or she can jump from volume to volume and create a web of associations. (Traditionally, there is even a certain cachet to knowing much of this library by heart: "Real Talmudists don't use CD-ROMs.")

This alternative rhetoric is exhilarating, and even somewhat visionary. It anticipates a populist era in which anyone with the right tools can tell you how many times (and where) the Bible mentions, say, "wine," or how many of Shakespeare's plots involve sending a letter, or which telephone numbers are unassigned in a given area code, or how to save a run-time version of a presentation, or what the odds are for this week's football games. They even expect that readers may pursue all these questions and receive all these answers in the same reading "session," while never leaving their bookless desks.

In this new communication culture, readers have nearly endless freedom in modifying their reading displays and in printing their personally constructed version of what they have read. With my own hypertext version of the Bible for example, I can set the typeface, the font size, and the column width; I can look at one version at a time, or shrink the panels to allow two or more versions at the same time—including Hebrew for the Old Testament and Greek for the New. I can export any subset of verses into my word processor, and if I think some of the commandments need changing, I can *edit them*!

Like everyone else who works with Microsoft Windows 3.1, or comparable systems, I have extraordinary powers of searching, sharing, and linking data. But I also have the power to produce unreadable, badly-designed displays and pages that would make the talented writers at Microsoft and Microsoft Press cringe! If, for example, I "maximize" my "Help Window," I can generate rows of text with 120 characters. (If I tweaked my setup a bit, I probably could get 200.) Or, alternately, I can produce a column so narrow, or a window so small, that almost nothing can be read without scrolling.

In the 90s, then, the author creates a library of molecules and modules of text and picture (not manuals and reports), and the readers—aided by linking, searching, and display tools—manufacture their own versions of the message. The sender remains rhetorically neutral; the readers persuade and instruct themselves.

This, say the enthusiasts for the alternative rhetoric, is really what readers and audiences have always done anyway. Learning and persuasion only occur when receivers *instruct and persuade themselves,* we often hear. No one reads the sentences in a technical publication in order. Now, though, the communicators and senders embrace the idea, instead of imposing irrelevant notions of speech. Now, writers and readers collaborate to do away with ordinary publications, which, they argue, tend to inhibit and frustrate all but the most scholarly readers' treks through hyperspace.

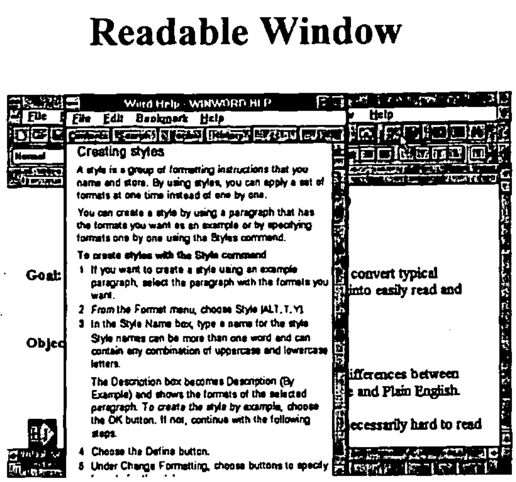


Fig. 1. Readable window.

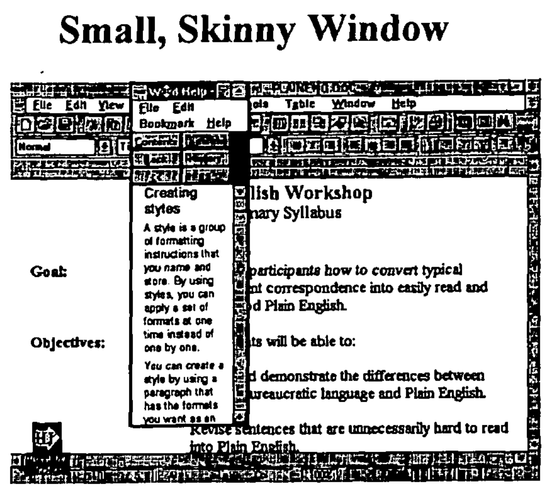


Fig. 3. Small, skinny window.

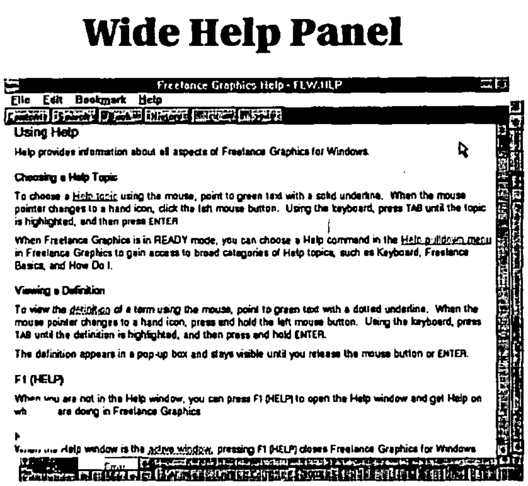


Fig. 2. Wide window.

SGML

Among the most interesting things about this new rhetorical culture, which often has a futuristic or even "New Age" cast to it, is that it has found an important ally in, of all places, the military. For quite different reasons, the Department of Defense (DoD) is also generally opposed to publications in which the document is linked to a particular piece of software and printer driver. Put simply, the military wants all future contractors to provide documents that are "compliant" with its CALS program (Computer-Aided Acquisition and Logistic Support) [7]. Instead of paper publications, instead of WordPerfect files, the military wants *SGML* (Standard Generalized Markup Language) documents. It is not only a broad approach to publication management, but also a specific ISO standard, and a Department of Defense standard as well [8]. SGML breaks the connection between a document and the software and hardware associated with the creation of that document. It reduces the document, no matter how it was created, to ASCII text and requires that all formatting information be rendered as ASCII tags, typically inserted between the < > brackets.

By severing the link between the document and its software and hardware, SGML lets publications be output to any printer . . . or screen, or speech synthesizer, or automatic translator, or archive storage device. Its codes and tags identify the logical and structural components in a document. But, unlike the codes familiar in word processing systems, SGML tags are NOT supposed to refer to specific typefaces, fonts, sizes, margins, tabs, leading, text highlights, page sizes, etc.

Instead of inserting <TimesRoman 24><center><bold> before the main title of an article, the SGML author inserts <maintitle>. Elsewhere, an output device consults a dictionary of Document Type Definitions (DTD's) and decides what combination of screen or printer instructions should be attached to "maintitle." These DTD's, written generally by someone other than the author, include formal definitions of document components and elements (using data dictionary logic) and hierarchical rules for what components may follow others.

The Dictionary of DTD's contains not only the current set of rules for the typefaces and indentation of headings; it also contains logical rules that prevent an author from skipping a level of indentation (for example, by going directly from <2ndhead> to <4thhead> without the intervening step). It also establishes scores of other precedence rules, such as forbidding a footnote in a caption or superscript in a footnote—if that's what the DTD developers wish.

So, curiously, the practical, administrative goals of the SGML/CALS enthusiasts dovetail nicely with the humanistic and liberal goals of the hypertext rhetoricians. Eventually, for example, the Navy wants to be able to download from satellites recent technical information from the files of its contractors to the workstations on its ships—instead of needing to carry 5 tons of manuals on a 90-ton vessel. How remarkably similar to what Ted Nelson wants for his Xanadu network! [9]

IMPLICATIONS FOR THE PROFESSIONAL COMMUNICATOR

There is a kind of fateful inevitability and technological elegance to the hypertext-document database-SGML trend. There is no point in resisting or opposing any trend attractive to both the military and the mystical! Indeed, the new technologies will probably create thousands of new, highly skilled jobs for technical communicators.

But during this transitional time, as the old rhetoric of sender-controlled writing yields to the new, we might ask ourselves a few technical and philosophical questions.

Does the writer/publisher of a document have an obligation to reduce the reading burden for ingenuous readers? That is, when we give readers the option to display and print our text in any way they like, should we be content with their sometimes unwise choices? Is it condescending to observe that many users of computers, especially those who use multi-panel screens, make poor choices: absurd and stressful colors; hard-to-read typefaces (when more readable ones are near at hand); counterproductive column widths, and so forth. If many of our readers are ingenuous about readability and page design, should we not protect them from themselves? If we know that badly designed pages and displays lead to fatigue, errors, and waste, can we encourage just such pages? Must we, like good citizens in a democracy, accept the right of readers to be wrong?

Can large publishing institutions be trusted to choose enlightened publishing standards? If we turn over our documents to our clients with only SGML tags, can we trust them to choose appropriate DTD's? Is there any historical evidence, for instance, that the federal government's principal publishers—like DoD or NTIS—appreciate the importance of typography and page design? Do we, as authors, have no right to demand that our articles and books look clear, accessible, and professional?

Can writing be an intellectually, professionally rewarding occupation if we break the traditional rhetorical link between the writer and reader? What professional writer would choose a career of putting thousands of chunks of information into a database stew? What pleasure is left in the process without a sense of the audience, the feeling that I am "speaking" words of my choice, in an order of my choice, to a person I recognize?

Technical communication is not a literary enterprise. In most cases, manuals and other publications must be produced by teams, with the individual contributors sacrificing their writerly egos to the overall project. Moreover, technical publications are hardly ever complete, and like computer programs, they are often modified and "maintained" by other writers. Indeed, it appears that manuals written entirely by one person—a method rarely used in sophisticated publication organizations—may be problematical because they are inherently unmaintainable. Even under these conditions, though, most professional writers still want to affect their readers in predictable ways, to practice their hard-learned craft, and to point to some products or creations that are more actual than virtual. Absent these small satisfactions, it is hard to imagine anyone wanting to be a technical writer at all.

REFERENCES

[1] M. Coney, "Technical readers and their rhetorical roles," in *IEEE Trans. Prof. Commun.*, vol. 35, pp. 58-63, 1992.
[2] E. Weiss, "Usability: Toward a science of user documentation," in *Computerworld, In Depth 9-16*, Jan. 1983.
[3] E. Weiss, "Ten Ways to Write an Unclear Instruction," in *Data Training*, June 1983.
[4] W. Strunk and E. White, *The Elements of Style 3/e*. New York: Macmillan, 1979, ch. 2.
[5] E. Weiss, *How to Write Usable User Documentation 2/e*. Phoenix: Oryx Press, 1991.
[6] D. Felker *et al.*, "Typographic Principles," in *Guidelines for Document Designers*. Washington, DC: American Institutes for Research, 1981.
[7] *Military Handbook, Department of Defense Computer-Aided Acquisition and Logistic Support (CALS) Program Implementation Guide*, MIL-HDBK, Dec. 1988.
[8] SGML is defined in ISO 8879 and Department of Defense MIL-M-2801.
[9] Ted Nelson, *Literary Machines*, Ed. 87.1, Self-published, 1987

Edmond H. Weiss, Ph.D. is an independent writer, lecturer, and consultant who travels North America teaching seminars on business and technical communication. He is the author of *One Hundred Writing Remedies: Practical Exercises for Technical Writing* (Oryx Press 1990), *How to Write Usable User Documentation* (2nd edition, Oryx Press 1991), and *The ISO 9000 Quality Manual: A Handbook/Workbook* (Edmond Weiss Seminars 1993). His base is Cherry Hill, NJ.

Of document databases, SGML, and rhetorical neutrality

Weiss, E.H.

This paper appears in: **Professional Communication, IEEE Transactions on**
Publication Date: Jun 1993
Volume: 36, Issue: 2
On page(s): 58-61
ISSN: 0361-1434
References Cited: 9
CODEN: IEPCBU
INSPEC Accession Number: 4484829
DOI: 10.1109/47.222682
Posted online: 2002-08-06 18:42:41.0

Abstract
New technology has enabled the audience to shape a writer's message. Today, publishing technical information often consists of letting the receivers search the files, extract what they judge relevant, sequence and organize it any way they wish, and even print or display it to their own specifications. Often, the writer is not creating deliberately worded and presented messages but rather, feeding molecular articles to rhetorically neutral databases, from which readers may extract what they wish. Such technologies as SGML even further limit writers and deprive them of such basic presentatic devices as deciding where pages will begin and end. The rhetorical implications of technology that empowers readers ar enfeebles writers are reviewed

Index Terms
Inspec
Controlled Indexing
electronic publishing full-text databases page description languages technical presentation

Non-controlled Indexing
SGML articles audience clarity cogency document databases file searching information extraction information organization new technology persuasiveness presentation publishing relevance rhetorical neutrality technical information writer's message
Author Keywords
Not Available

References

No references available on IEEE Xplore.

Citing Documents

No citing documents available on IEEExplore.

Using SGML to create complex interactive documents for electronic publishing

Commentary

—PETER GOLDIE

Abstract—*In creating complex interactive documents, some technical communicators use software products that emphasize format and style in displaying pages. This approach limits the communicator's ability to repackage the information presented in electronic versions and increase its interactive use, which is a key benefit of the structure-based approach offered by using Standard Generalized Markup Language (SGML). In a number of projects that render mathematical, scientific, and engineering texts electronically, using SGML allows the technical communicator to make equations interactive and to automate links to references. The author sketches out problems associated with page description approaches to displaying electronic pages and discusses the comparative benefits of SGML.*

Index Terms—*Electronic publishing, complex documents, SGML, page description.*

Manuscript received November 1996; revised February 1997.
The author is with Lightbinders, Inc., San Francisco, CA 94107 USA.
IEEE PII: S 0361-1434(97)04360-9.

As ONE encounters new media products, it is unusual when a product is NOT advertised as interactive. *Interactive*? How could anything used by humans not be *interactive*? Books are interactive, TV is interactive, radio is interactive—their designs all require a human interaction. Do the publishers really mean nonlinear? Do they mean integrated with multimedia? Do they mean the contents can be modified?

Complex, on the other hand, is a word we rarely see in any new media product advertisement. The word *complex* is a kiss of death when describing any computer product intended for the general public. In the "new world" of the Information Age, everything is supposed to be easy to use and understand; especially if in the "old world" it was *complex*.

Let us begin by defining the terms *complex document* and *interactive document* in the context of electronic publishing:

A *complex document* is a highly evolved means of textual communication, coming in a wide variety of formats agreed upon by convention and guided by numerous style manuals. A *complex* document has a hierarchical structure that results from methodical analysis of the contents, as applied through the conventions of presentation style.

Inherent in any complex document structure is navigation, or knowing where you, as the reader, are in relation to the total contents. In the sim-

plest of navigation schemes, pages are numbered; in more complex documents, tables of contents (TOCs) outline the structure of frontmatter, chapters, subchapters, and backmatter. The large structural features of a complex document are further subdivided into sections, subsections, section titles, sublevel headers, paragraphs, lists, tables, figures, appendices, indices, and so on. Many major scholastic fields claim unique structural features are necessary to properly communicate their specific type of information.

Interactive documents have some or all of the following characteristics.

- Multimedia manipulation, including images, animation, audio, and video: Integration of visual or audio components into publications is not new, but the reader's ability to zoom and pan images, play back video with fast-forward, pause, and rewind, and visually examine an audio spectrum histograph during sound generation increases user interaction.
- Nonlinear use, alternative pathways: Most print publications are linear, with fixed results or conclusions. Nonlinear pathways through complex documents are possible, i.e., tailoring distinct lesson plans out of linear sequence from within a course textbook. Although this is possible with print products, electronic titles can instantly and virtually reorder the contents by design. Alternative pathways could lead users toward different perspectives or conclusions.
- Evoking static and nonstatic external actions: External actions can be launched from within a complex document. These actions can be as simple as displaying an electronic image, to as complex as operating machinery in a remote location. Display of a fixed image is a simple static action; activation of a program that requires user input at decision points would be a nonstatic action.
- Ability to modify, annotate, or reorder the contents: By definition, an interaction requires actively changing the original conditions. The contents of a "totally" interactive document can be changed, edited, redlined, annotated, and rearranged by the reader.

Document Structure vs. Style

The majority of electronically published products fall far short of being able to deliver complicated documents in all of their richness, let alone documents that are both complex and interactive. The reason behind this limitation is directly related to the evolution of word processing on personal computers (PCs, be they Mac or "IBM"). The technology of word processors arose as a replacement for typewriters (specifically, the IBM Selectric II) with the design goals of achieving a hardcopy output of typescript quality. At the same time, printers had already adopted very complex, very expensive, very proprietary computerized typographic composition systems. As PC hardware and software capabilities grew, the desktop publishing industry arose, first presenting itself as a naïve challenge to printing. Ironically, while never a real challenge to large press operations, DTP provided *de facto* standards upon which designers and production editors using PCs could create a formatted page that printers could use to produce the final hardcopy results.

When a complex document is delivered as print, the style alone indicates the complex nature of the document structure. Style conventions such as typeface, font, point size, bold/italic/underscore emphasis, layout, tabs, indents, and rules give the reader the visual clues necessary to convey the document structure. Style is, in fact, a secondary characteristic of the underlying document structure. Because style conventions are traditionally predicated upon the layout of a printed page, electronic publication systems which rely upon page descriptive languages are designed to communicate structure through presentation style, giving page layout priority over document structure. The most common example of this approach to rendering electronic documents is Adobe Postscript Language, and its electronic publishing descendant, Adobe PDF/Acrobat. See Fig. 1 for an example of a PDF page.

The underlying, and unfixable, problem with Postscript/PDF is that it is a page-descriptive language, not a document-structure-descriptive language. Style and page layout appearance take precedence over document structure. Any complicated document can be rendered in Postscript, but the structure must first be interpreted by a human operator into the appropriate style, thus communicating the underlying structure through presentation style. Unfortunately, once a complex document has been prepared as a Postscript/PDF file, there is no reliable, accurate, and/or automated process of applying a logical structure back to the document, despite some rather ingenious attempts [1].

If a document could contain structural information encoded to be machine-readable, style could then be applied automatically. If an author writes his manuscript within a structured framework, the process of organizing, editing, revising, indexing, referencing, and, of course, hardcopy generation, can be greatly simplified. This is the fundamental reasoning behind the area of Standard Generalized Markup Language (SGML) for complex documents.

Many other benefits also come from structured document encoding. Although SGML originally began as a proprietary IBM format called GML, its authors recognized its widespread potential and proposed the development of an international, platform-independent, open, "standardized" GML. This it became, as

the ISO-8879 standard. For more of the history behind SGML, and other online SGML resources, see several excellent web sites that provide considerable information [2]–[6]. An open text encoding standard allows publishers to set common document structure conventions and allows any software developer to create applications that can apply the standard. Repackaging contents into new titles is simplified because source materials are organized by document structure, not by page. Archiving the contents is aided by platform-independence and a delimited data structure readily adaptable to fielded databases. Rendering SGML structure becomes a straightforward matter of assigning style to a structure element. Changing document appearance (style) becomes simple and instantaneous; one simply applies a new set of styles (stylesheet) to the underlying structure.

SGML is a language for describing the structure of types of documents and the text encoding to be used in creating content consistent with those document types. Any individual, publisher, or industry is free to create its own document structures, called document type definitions (DTDs), and these can be either public domain or proprietary. Thus in SGML, publishers are given a logical, open, extensible, customizable, interconvertable, indexable, and archival text encoding system. Certain federal government agencies, especially those that must exchange large volumes of technical documentation with industry (i.e., military, transportation departments), have mandated SGML formatting. Publishers, for the most part, have all endorsed SGML in concept, but only a relative few have committed their production to SGML. The reason for slow adoption of SGML is economic and technical. More on this later.

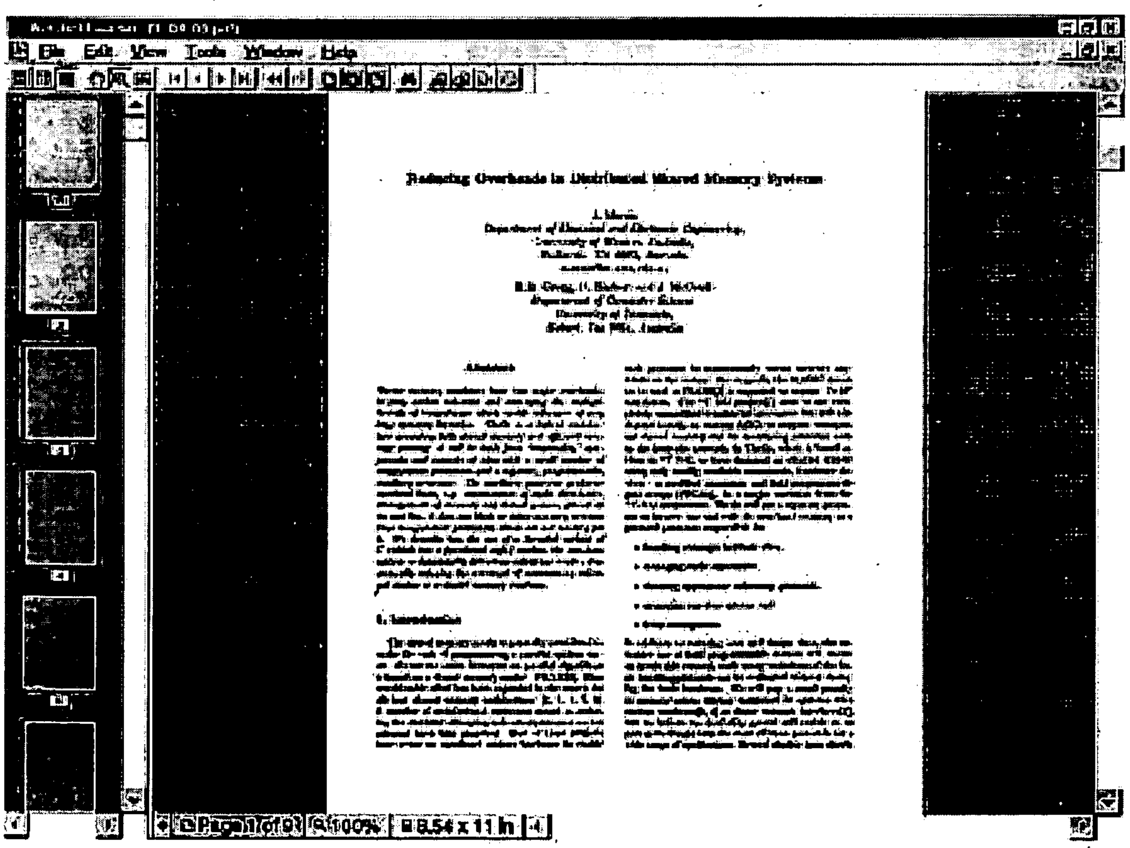
Why We Chose Structured Document Encoding

In 1988, I created Lightbinders to produce CD-ROM versions of basic research publications for nonprofit professional societies and commercial science, technology, and medical publishers. The initial technical demands we were required to solve included the following:

- documents must include full text, with searchable index;
- documents must include black and white, halftone, and color illustrations;
- documents must work on PC-DOS and Macintosh platforms.

These minimal requests were difficult to fulfill in the late 1980s-early 1990s, especially considering the level of installed hardware. But the hardware, operating systems, and user knowledge rapidly changed, and as the readers of the electronic ver-

Fig. 1. An IEEE journal article rendered through Adobe Acrobat, using a page descriptive language, Postscript/PDF. Presentation and navigation is inherently limited by the layout of the printed page.



sions became more sophisticated, they demanded:

- Same appearance and style as the print version;
- Many special characters (foreign, math, chemical symbols);
- Navigation aids, bookmarks, tables of contents;
- Search forms, history, hit lists;
- Hypertext and hyperlinks;
- Support for more platforms (MS-Windows, Macintosh, UNIX);
- Image manipulation, zooming;
- Audio/video playback;
- Integration of external software (molecular modeling, interactive mathematics).

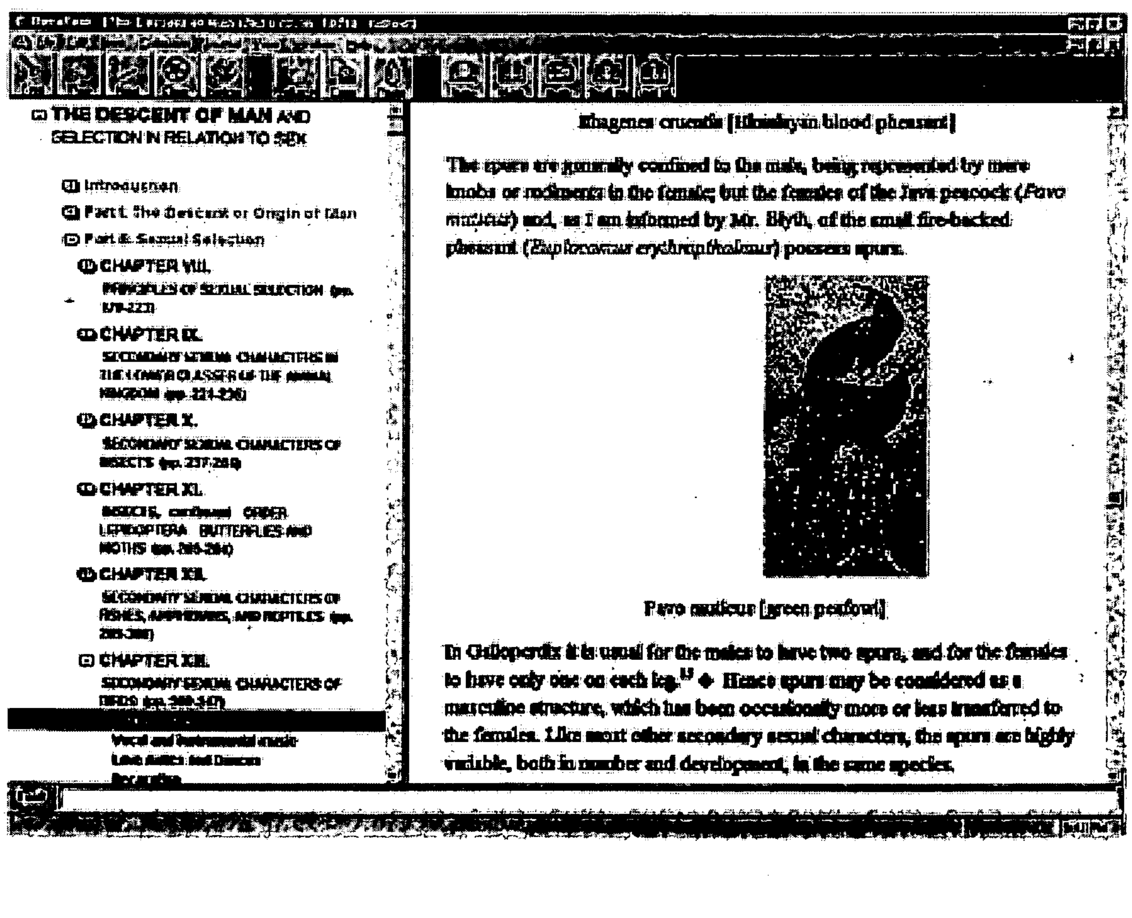
It was clear that we needed a better cross-platform delivery system, one that would use an underlying encoding system that would survive migration of the contents into the next generation of software. This we found in the *Dyna*Text Publishing System created by Electronic Book Technologies, EBT (now a division of INSO Corporation [7]). At the time, *Dyna*Text was the only software interface using native SGML encoding that met the technical demands of our clients, the scholarly publishers.

Even though we were a small organization, the prospect of training the staff accustomed to flat ASCII to code and edit SGML appeared to be a huge challenge. There were almost no editing tools available, SGML consultants were expensive and lived in distant cities, and DTDs appeared incomprehensible in their brevity. Fortunately, looks were deceiving.

SGML tagging proved to be entirely logical and easy to learn. All document contents are contained within elements or tags; these tags are symbolized as simple abbreviations inside corner brackets. Paragraphs are nested within subsections, subsections within sections, sections within chapters. Whereas a typesetting composer needed to take a structural element, such as an article title, and apply many aspects of style to it (font, point size, color, emphasis, positioning, and so on), an SGML editor simply has to code `<title>Article Title...</>`, and confirm the title element is located correctly within the larger structure of the document (i.e. inside the `<fm>frontmatter</>`). Separation of style from structure actually improved our editing productivity. Fig. 2 below shows a page from a document produced with SGML.

The multilevel table of contents is automatically created from the SGML document structure, allowing dynamic navigation to the text. Scalable text automatically wraps to any window size; alternative presentation styles (fonts, colors, layout, etc.) can be selected by the user. Hyperlinks and hypertext embedded within the SGML source are indicated by colors and icons.

Fig. 2. A simple SGML application, as viewed through the *Dyna*Text interface.



One of the software tools that has enabled *Dyna*Text to be the premier SGML browser is the stylesheet editor, *Insted*. This software application presents the contents organized within the document structural hierarchy, and it allows style to be individually applied to each element type. The style of each element is described by a set of style properties, which can be set by property value functions. The variable properties are such things as font type, font weight, font slant, font size, color, horizontal and vertical spacing, and preceding and following text for each instance of a given element. Property value functions can be used to vary the values of properties programatically, based on operating system information, mathematical operations, string operations, and information about the contents, attributes, and hierarchical position of elements within the document. A set of these style definitions is referred to as a stylesheet. *Insted* applies property value functions to properties interactively.

By using *Insted*, the technical communicator can easily tune stylesheets while judging their effect on screen, which is functionally somewhat similar to formatting onscreen with advanced word processors. Stylesheet editing produces the computer display appearance; tables of contents are automatically extracted; and hardcopy options are created. Stylesheets can also be taught to monitor their environment, to automatically select the correct and best font for each operating system, to scale the type correctly for different monitor resolutions, and to automatically render illustrations and tables to fit the window dimensions.

Stylesheet editing through *Insted*, in addition to styling, gives the content developer control of the "actions" related to an element. Through a simple script, one can automatically generate page numbers, chapter numbers, either as digits, letters, Roman numerals, or combinations thereof. Illustration tags are scripted to instruct the *Dyna*Text browser to open the graphic files. Elements for external functions can be embedded anywhere in a document, and through scripting, they can launch external programs. We created element tags such as `<video>A Video Clip</>`, which, when clicked on, play a video segment. Any tagged element can be directed to evoke an action.

In our work for outside publishers, we are often directed to use their "standard" DTD. Ideally, their DTD has been created with consideration for all document structure and functionality found within the content they wish to publish. It is not uncommon for us to encounter limitations in their DTD, and we are usually able to quickly modify the DTD to include the new element definition. In addition, *Dyna*Text is somewhat tolerant of violations to strict SGML structure, and this we occasionally use to our advantage. When a needed tag did not exist in the DTD we were directed to use, we can often simply use a new tag name without modifying the DTD.

SGML and Mathematics

EBT has wisely incorporated a public domain version of TeX (emTeX) into their publishing system as a means of rendering mathematics. SGML still lacks a universally agreed-upon method of encoding mathematics [8], whereas TeX on its own (actually, thanks to the huge efforts of Donald Knuth [9]) has become a *de facto* standard for rendering math notations. The ability to integrate TeX, which is a programming language that has no direct relationship with SGML, provides a clear demonstration of the versatility and extensibility of SGML. TeX can be contained inside an element we called `<tmath>`, as in this example:

`<tmath>$E = mc^{2}$</>`.

During the process of compiling the SGML into a book, we direct any raw TeX within `<tmath>` to be run through the emTeX compiler, resulting in a renderable DVI (device independent) object. The DVI object is embedded in the compiled SGML, and through an *Insted* program script, we direct embedded `<tmath>` DVIs to be viewed as rendered DVIs. The emTeX DVI calls Computer Modern math fonts (the fonts are part of the *Dyna*Text browser installation). The result is dynamic rendering of any math formula we have encountered, and this enabled us to produce the CD-ROM version of the classic math text, *Table of Integrals, Series, and Products*, by Gradshteyn and Ryzhik (Academic Press; ISBN 0-12-294756-8).

This is not the limit of how we make mathematics more interactive though SGML and TeX. Frequently the TeX source would be valuable to readers who write in LaTeX, AMS-TeX, or one of the other TeX's. We duplicate the source in a nearby element called

`<rawtex>$a^2 + b^2 = c^2$<\>`

and suppress default display of `<rawtex>` behind an icon or through a stylesheet element *hide* command. When revealed by the reader, the source TeX can be cut and pasted into his TeX application, saving considerable time rewriting the source notation.

We also use TeX displays to launch from the SGML content into external interactive mathematics programs, such as *Mathematica, MathLab, MathCad*, or *Maple*. Here, the rendered TeX math element is scripted to represent a hot spot; clicking on the math formula would start the math program, pass the formula to be interpreted, and await input of new variables from the reader. This we demonstrated in a handbook on acoustical research: a TeX formula illustrated the basis for calculating sound frequency of a vibrating object; clicking on the formula launched the external program, allowed the user to change the physical shape of the object, and automatically regenerate the new frequency ... and hear the new frequency as well!

SGML and Hyperlinks

Although the use of SGML provides a means to represent the structure of a complex document, and textual documents tend to be read in a linear manner, SGML can easily be adapted for nonlinear or alternative pathways through contents. In fact, traditional documents already employ some nonlinear aspects, such as footnotes and reference citations. SGML enables tagging of these cross-reference points, and program scripting directs the hyperlinking of the spatially separated sections. Clicking on a reference citation instantly brings up a citation window from the bibliography. This does not mean that every reference must have the citation embedded at that spot (hypertext), just that a simple but unique code is added to associate the reference with its citation. Here is an example of the simplicity of hyperlink coding in SGML:

Reference coding:

... and these results were confirmed by `< ref id = R1234 > Smith et al., 1984 </>`. In 1993, we ...

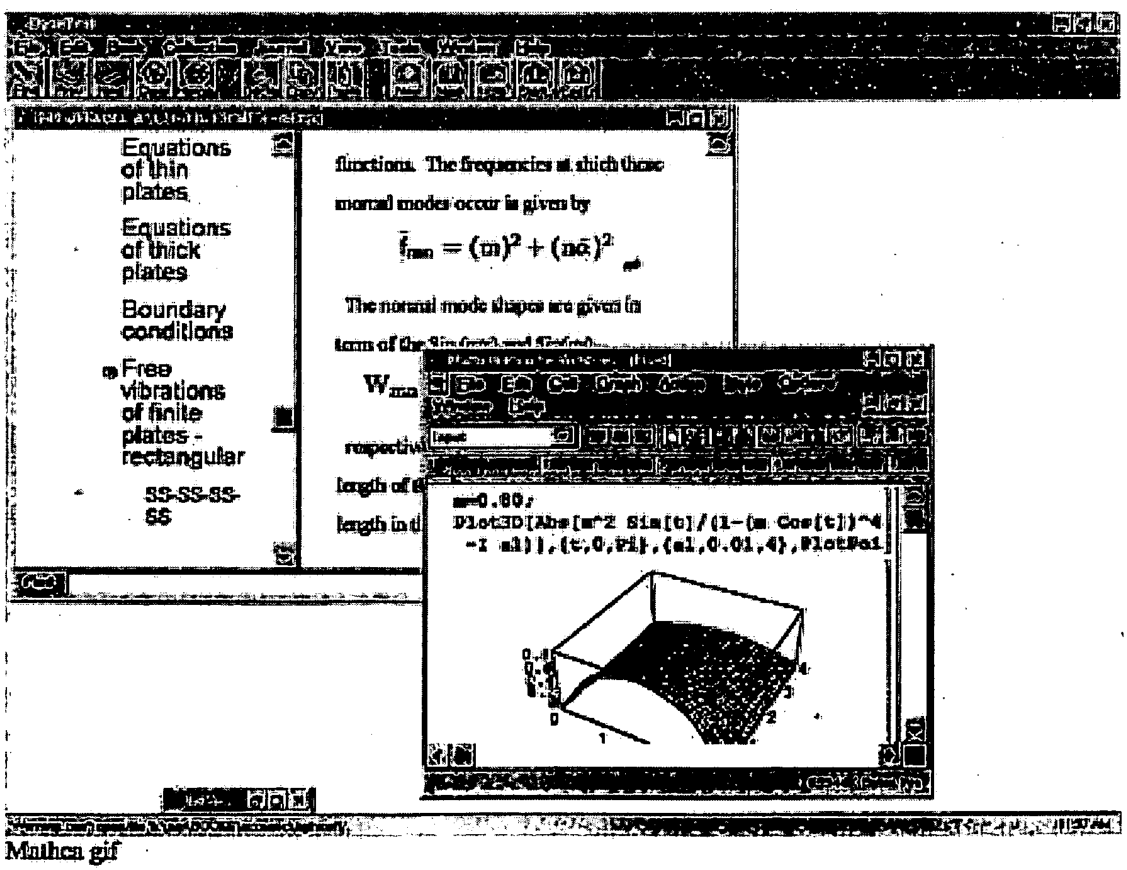
Citation coding:

`<bib id = R1234>Smith, A.R., Jones, R.T., and Royce, P.M. The Evolution of Diacritical Thought. 1984, W. H. Nueroth & Sons.</>`

Through simple programming scripts we associate with the element `<ref>` and `<bib>` tags, and the *Dyna*Text application can match the unique ID codes and reveal the linked text within a new window.

The addition of hyperlinks has become so routine in our SGML production that we routinely link all bibliographic references, as well as author, footnotes, figure, table, sidebar, appendix, and references. Because of the structured nature of SGML, the task of locating link sources and targets can be automated. This is because the target of a link is a defined element, to the exclusion of all other elements. The hierarchical structure allows the communicator to unambiguously address elements within the hierarchy such as, the parent's left sibling. This allows for such things as restricted scrolled popup views. One CD-ROM we produced (*Methods in Enzymology Index CD-ROM 1955–1994*, Academic Press, ISBN 0-12-000101-2) contains over 320 000 hyperlinks, proving that the process of hyperlink coding can be performed economically, and frequently can be automated. Embedding of such high numbers of hyper-

Fig. 3. A math formula in *Dyna*Text is linked to launch a Mathematica notebook by clicking on the red arrow icon. The Mathematica notebook application can allow the reader to interactively alter the formula parameters and view the results.



links in Acrobat titles has not been found to be practical.

More elaborate examples of nonlinear pathways through SGML can be achieved through stylesheet editing. Sections of contents can be exposed or hidden selectively, giving the reader the impression of re-ordered contents. Finally, a feature of the *Dyna*Text browser lets the reader record his own movement through the contents—forward, backward, or across different titles. The individual path the reader has followed can be saved, edited, and played back. This is an excellent method for teachers to prepare course lessons from within a large textbook. To navigate, this feature depends upon the underlying SGML structure of the document as the software interprets each structural element. This precision is not possible using a page descriptive language, where the structural unit is an arbitrary page. Adobe Acrobat is only able to achieve a certain level of navigation by creating text-based indicies in parallel to the PDF representations of each page. The added cost of integrating searchable and navigable text to PDF files is significant.

The ultimate in complex document interactivity comes when the contents can actually be changed by the "reader" (obviously, the definition of "reader" would now encompass "writer/editor"). When a document is structured, and presentation of the contents is dynamically rendered each time it is displayed, any changes made will automatically be seen. In a page descriptive language, each page is dependent upon the prior page. Addition or deletion of a paragraph on page 1 would require all subsequent pages to be recreated. The distillation process of Postscript to PDF produces output pages that are fixed and unalterable. The latest release of *Dyna*Text ver. 3.0.1 is capable of rendering raw SGML, thus allowing a "reader" to change raw contents and immediately see the results.

SGML's Missing Links

All of the benefits of SGML are dependent upon SGML-aware software tools—tools for authoring, editing, styling, indexing, presentation, archiving. Development of these tools has been slow, and development of applications for the two most critical junctures of the publication process, authoring and printing, have been slowest of all. Only two significant word processors have been created using SGML as their primary format, **SoftQuad** *Author/Editor* and **ArborText** *Adept Editor*. Both of these applications cost many times what common word processing applications cost. There are add-on SGML modules to the popular word processors *Word* and *WordPerfect*, but these add-on modules must function within the constraints of the parent application and its proprietary format, and thus they have limitations.

Without an inexpensive SGML-authoring word processor, publishers continue to receive manuscripts in any word processing format and must convert the author source to their internal editorial format. Following editorial review and approval, the accepted manuscript often must be converted into the typographic composition format, copy-edited for style, and sent to press. As producers of the SGML-based electronic version, we must convert whatever format we receive from the publisher or printer to SGML. When SGML is introduced "downstream" into the publication process, the conversion/translation costs can be high. A commitment to SGML throughout the entire publication process would result in better products, substantial savings, and improved speed of publication.

On the print side, a publisher that can generate SGML finds few press systems able to receive SGML encoding. Typesetters and printers have major infrastructure investments in hardware and software designed to produce hardcopy. They also recognize that SGML is a serious threat to their businesses. Those printers using Postscript to drive their presses can produce PDF files for their publisher customers at little added cost. By providing an inexpensive and expeditious electronic product in Adobe Acrobat to their customers, printers have given publishers a compelling reason to delay more widespread adoption of SGML.

SGML on the Web

Despite a logical developmental path to include web browsing, *Dyna*Text remains a CD-ROM/LAN browser. EBT has elected to develop the SGML server side of the Internet publishing solution, avoiding the contentious "browser wars." EBT's SGML server, called *Dyna*Web ver. 3.0, accepts the SGML content prepared for CD-ROM/LAN delivery and serves it as HTML-acceptable to existing web browsers such as Netscape Navigator and Internet Explorer. The result is a quite respectable interim solution, even with the limited capabilities of existing Web browsers (see Fig. 4). SoftQuad [10] has developed *Panorama Pro* ver. 1.5, an SGML browser that commonly functions as a helper application launched from within a web browser. The SoftQuad system requires all SGML source files (SGML content, DTD, stylesheets, entities, graphics) from each document to be sent before rendering online. Viewing, linking, and searching is then limited to the one document being viewed. This system does not include a means of indexing, searching, and viewing a collection of documents as a whole.

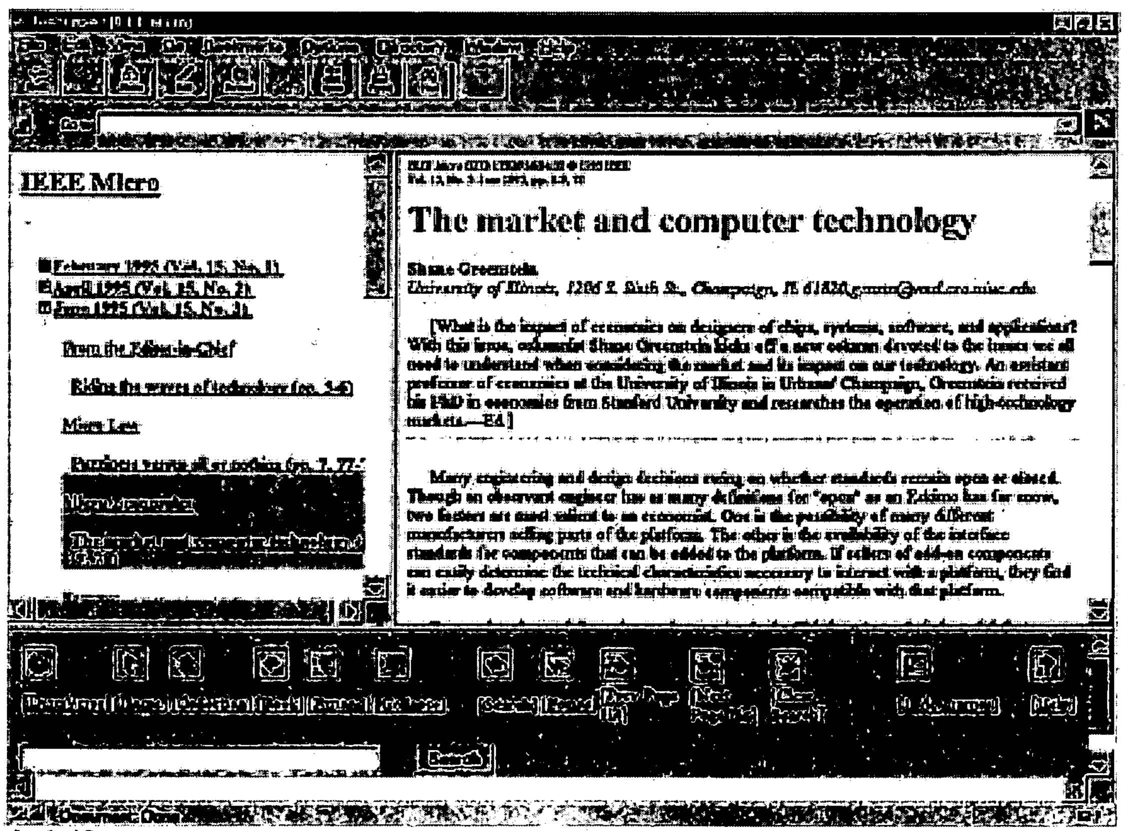
The World Wide Web has provided readers a huge sampling of different electronic publication schemes, including thousands of examples of Acrobat. Most readers browsing online, however, only encounter HTML, the "language" of the Web. Many people readily recognize a compelling, easy to program, and simple to use format in HTML, yet never realize that HTML is a simplified application of SGML. In his original design of the World Wide Web, CERN scientist Tim Berners-Lee saw the bene-

fits of adopting an established, non-proprietary, platform-independent text encoding system, readily capable of extensive hyperlinking. HTML as a simple, stripped-down application of SGML, can deliver some document structure. Newer versions of HTML are now in the process of reconstituting the richness of its SGML origins. A more logical approach to improving document delivery over the Web would be to stop improving HTML (to function within existing Web browser capabilities) and instead advance Web servers and browsers to full SGML capabilities.

Recent developments help illustrate why SGML will likely be the next logical step in the evolution of the World Wide Web. The ability to modify and customize DTDs to suit individual publishers' requirements has resulted in many types of valid SGML. Common Web browsers (Internet Explorer, Netscape Navigator) have been designed for one simplified SGML application, HTML, and were not created to handle the variety of DTDs. One possible solution is the proposed XML (eXtensible Markup Language) standard, which will enable publishers to quickly port their existing SGML content to the Web without "dummying" it down to the level of HTML. XML was developed by an SGML Editorial Review Board formed under the auspices of the World Wide Web Consortium (W3C) in 1996 and chaired by Jon Bosak of Sun Microsystems, with the very active participation of an SGML Working Group also organized by the W3C [11].

By smoothing the road to getting SGML online, *access* to contents is improved, but not *awareness*. The explosion of Web contents points to the most important reason to use a structured markup format. The process of online publishing provides access to materials, but without cross-collection searching, most users cannot locate what they need to find. While the popular Web searching tools (such as Yahoo, Alta Vista, and Lycos) provide some means of narrowing a search, they offer rather poor accuracy compared to traditional library search methodology. Bruce Schatz, Principal Investigator of the Digital Library Initiative project [12] at the University of Illinois, and his colleagues have addressed the related problems of searching collections distributed across the network and relating the jargon of specialized fields through semantic indexing. Toward this end, they have created a testbed of a major engineering digital library based

Fig. 4. An example of SGML-based IEEE journal delivered through an HTML internet browser (Netscape Navigator), via the *Dyna*Web server. This system provides structured searching, hyperlinks, and other interactivity made possible by SGML encoding.



dwebgif

upon SGML documents [13]. Complex and structured searching aimed at accurate retrieval from large data repositories will require structured documents.

Summary

The lack of critical SGML software applications has enabled Adobe Acrobat to become one of the most common formats for electronic publication online. The low cost of PDF production, however, is increasingly becoming less important as publishers recognize the greater long-term benefits of SGML-centric production. More important than publisher recognition of SGML is reader (market) acceptance. Acrobat's origins as a page-layout format place severe restrictions on its appearance onscreen, the quality of illustrations, and the speed of transmission. Online readers have come to expect text wrapping to fit the available window, powerful searching, extensive hyperlinking, and high-quality graphics. Acrobat fails to deliver these critical features.

If the goal of electronic publishing is complex and interactive documents capable of multimedia, hypertext, hyperlinks, dynamic rendering, unlimited choice of fonts and characters, control over presentation online and print, advanced searching, repackaging/republishing potential, and a nonproprietary archive format, SGML provides a logical and very efficient solution. Adobe PDF/Acrobat advocates will argue that some of these features are possible in Acrobat; however, once one attempts to maximize the functionality of an Acrobat document with indexing, navigation, and hyperlinks, the short-term cost advantage quickly disappears.

Simply put, SGML is the "acid-free paper" of the electronic world. Despite its clear benefits, however, the limited acceptance of SGML in electronic publications is no mystery. The high cost of post-compositional translation of text into SGML and the resistance of typesetters and printers to retooling their considerable infrastructure are real disincentives to change. Adobe Acrobat has provided an alternative that is cheap and "good enough" for the moment, but the fundamental problems of being based on a page descriptive language prevent it from becoming a comprehensive, long-term electronic publishing solution.

What I do find hard to accept are software developers who fear to challenge the *de facto* publishing standards being established by major corporations. This will change, and all it will take is one brave and innovative company with an SGML-based "killer app." I can describe several killer SGML applications in detail today; all I wonder is who will build them, and when.

References

[1] W. S. Lovegrove and D. F. Brailsford, "Document analysis of PDF files: Methods, results and implications," *Electron. Pub.*, vol. 8, no. 3, pp. 1–14, 1995.

[2] SGML Open is a nonprofit, international consortium of suppliers whose products and services support the Standard Generalized Markup Language: http://www.sgmlopen.org/.

[3] The NCSA/SoftQuad SGML on the Web Page is a joint production of NCSA, the National Center for Supercomputing Applications, and SoftQuad Inc. It is maintained by Lucy Ventresca and the staff of SoftQuad.: http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/WebSGML.html.

[4] The SGML Centre provides consultancy and advice on the application of the Standard Generalized Markup Language (SGML) and related standards/applications: http://www.u-net.com/~sgml/.

[5] The Whirlwind Guide to SGML Tools and Vendors, maintained by Steve Pepper: http://www.falch.no/people/pepper/sgmltool/.

[6] The SGML Web Page maintained by Robin Cover: http://www.sil.org/sgml/sgml.html.

[7] Electronic Book Technologies (EBT), a division of INSO Corporation, is located at One Richmond Square, Providence, RI 02906. Tel: (401) 421-9550, Fax: (401) 421-9551, Email: info@ebt.com, Web: http://www.ebt.com.

[8] SGML and the Semantic Representation of Mathematics, Roy Pike, Clerk Maxwell Professor of Theoretical Physics, King's College, Strand, London, U.K. Stephen Buswell, Stephen Healey, Martin Pike, Stilo Technology Ltd., Empire House, Mount Stuart Square, Cardiff, UK. 11. April. 1996: http://mish161.cern.ch/sc4wg6/math/pike.htm.

[9] D. E. Knuth, *Computers & Typesetting:–Volume A, The TeXbook.* Reading, MA: Addison-Wesley, 1984, x+483pp. ISBN 0-201-13447-0.
——, *Volume B, TeX: The Program.* Reading, MA: Addison-Wesley, 1986, xviii+600pp. ISBN 0-201-13437-3.
——, *Volume C, The METAFONTbook.* Reading, MA: Addison-Wesley, 1986, xvi+451pp. ISBN 0-201-13445-4.
——, *Volume D, METAFONT: The Program.* Reading, MA: Addison-Wesley, 1986, xviii+566pp. ISBN 0-201-13438-1.
——, *Volume E, Computer Modern Typefaces,* Reading, MA: Addison-Wesley, 1986, xvi+588pp. ISBN 0-201-13446-2.
[10] SoftQuad is located at 20 Eglinton Ave. West, 12th Floor, P.O. Box 2025, Toronto, Ont., Canada M4R 1K8, Tel: (416) 544-9000, Fax: (416) 544-0300, Email: mail@softquad.com, Web:http://www.softquad.com.
[11] Extensible Markup Language (XML): W3C Working Draft 14-Nov-96, http://www.textuality.com/sgml-erb/WD-xml.html.
[12] The Digital Libraries Initiative (DLI) project at the University of Illinois at Urbana-Champaign is developing the information infrastructure to effectively search technical documents on the Internet. Their testbed digital library is based in Standard Generalized Markup Language (SGML) from engineering and science publishers: http://dli.grainger.uiuc.edu/.
[13] B. Schatz, "Information retrieval in digital libraries: Bringing search to the net," *Science*, vol. 275, pp. 327–334, 1997.

Peter Goldie received his doctorate from the Sackler Institute of New York University, specializing in the biochemistry and immunology of human malaria. In 1988, Dr. Goldie founded Lightbinders, with the implicit goal of developing electronic publication methods for scientific and technical titles. Lightbinders has produced dozens of academic CD-ROM and online titles, including *Journal of Biological Chemistry, Protein Science, Optics Letters, Methods in Enzymology,* and IEEE/Computer Society *Magazines* and *Transactions.* He is most proud of the *Darwin Multimedia CD-ROM, 2nd Edition,* which is the result of an ongoing international collaborative project now in its 8th year.

Access this document

Full Text: PDF (244 KB)

Download this citation

Choose Citation & Abstract

Download ASCII Text

[redacted]

» Learn More

Rights and Permissions

» Learn More

Using SGML to create complex interactive documents foı publishing

Goldie, P.
Lightbinders Inc., San Francisco, CA, USA;

This paper appears in: **Professional Communication, IEEE Transactions on**
Publication Date: June 1997
Volume: 40 , Issue: 2
On page(s): 129 - 138
ISSN: 0361-1434
CODEN: IEPCBU
INSPEC Accession Number:5611768
Digital Object Identifier: 10.1109/47.588825
Posted online: 2002-08-06 21:29:40.0

Abstract
In creating complex interactive documents, some technical communicators use software emphasize format and style in displaying pages. This approach limits the communicator's the information presented in electronic versions and increase its interactive use, which is structure-based approach offered by using Standard Generalized Markup Language (SG projects that render mathematical, scientific, and engineering texts electronically, using S technical communicator to make equations interactive and to automate links to reference sketches out problems associated with page description approaches to displaying electrc discusses the comparative benefits of **SGML**

Index Terms
Inspec
Controlled Indexing
electronic publishing interactive systems mathematics computing page descr languages technical presentation user interfaces

Non-controlled Indexing
SGML Standard Generalized Markup Language electronic publishing engine interactive documents interactive use mathematical text page description pε page style scientific text software products technical communicators

Author Keywords
Not Available

References

1 W. S. Lovegrove and D. F. Brailsford, "Document analysis of PDF files: Methods, resı *Electron. Pub.*, vol. 8, no. 3, pp. 1-14, 1995.

2 SGML Open is a nonprofit, international consortium of suppliers whose products and Standard Generalized Markup Language.

3 The NCSA/SoftQuad SGML on the Web Page is a joint production of NCSA, the Natiı Supercomputing Applications, and SoftQuad Inc. It is maintained by Lucy Ventresca ε SoftQuad.

4 The SGML Centre provides consultancy and advice on the application of the Standar Language (SGML) and related standards/ applications.

5 The Whirlwind Guide to SGML Tools and Vendors, maintained by Steve Pepper.

6 The SGML Web Page maintained by Robin Cover.

7 Electronic Book Technologies (EBT), a division of INSO Corporation, is located at On Providence, RI 02906. Tel: (401) 421-9550, Fax: (401) 421-9551.

8 SGML and the Semantic Representation of Mathematics, Roy Pike, Clerk Maxwell Pr Physics, King's College, Strand, London, U.K. Stephen Buswell, Stephen Healey, Ma Technology Ltd., Empire\ House, \ Mount \ Stuart \ Square, \ Cardiff, \ UK, Apr

9 D. E. Knuth, *Computers \& Typesetting:\--Volume A, The TeXbook*, <2emrule>, Volun Program.\ Reading, MA: Addison-Wesley, 1986, xviii+600pp. ISBN 0-201-13437-3. · The METAFONTbook.\ Reading, MA: Addison-Wesley, 1986, xvi+451pp. ISBN 0-20 <2emrule>, Volume D, METAFONT: The Program.\ Reading, MA: Addison-Wesley, ISBN 0-201-13438-1. <2emrule>, Volume E, Computer Modern Typefaces,\ Readin Wesley, 1986, xvi+588pp. ISBN 0-201-13446-2 Reading, MA: Addison-Wesley, 0-20

10 SoftQuad is located at 20 Eglinton Ave. West, 12th Floor, P.O. Box 2025, Toronto, O Tel: (416) 544-9000, Fax: (416) 544-0300, 2025.

11 Extensible Markup Language (XML): W3C Working Draft 14-Nov-96.

12 The Digital Libraries Initiative (DLI) project at the University of Illinois at Urbana-Chan the information infrastructure to effectively search technical documents on the Interne library is based in Standard Generalized Markup Language (SGML) from engineering publishers.

13 B. Schatz, "Information retrieval in digital libraries: Bringing search to the net," *Scienc* 334, 1997.
[CrossRef]

Citing Documents

No citing documents available on IEEE Xplore.

INTRODUCTION TO SGML[1] CONCEPTS

Paul Ellison

This brief paper provides an introduction to the concepts of SGML, as part of the set of papers that make up the IEE Event entitled 'Adding Value to Documents with Markup Languages'.

The Markup Process Text processing systems typically require additional information to be interspersed among the natural text of the document being processed. This additional information, called "markup", serves two purposes:

a) Separating the logical elements of the document, and

b) Specifying the processing instructions to be performed on those elements.

In publishing systems, where formatting can be quite complex, the markup is usually done directly by the user who has been specifically trained for the task. In word processors, the formatters typically have less function, so the (more limited) markup can be generated without conscious effort by the user.

It is therefore important to consider how the user of a high function system marks up a document. There are three distinct steps, although he may not perceive them as such.

a) He first analyzes the information structure and other attributes of the document; that is, he identifies each meaningful separate element, and characterizes it as a paragraph, heading, etc.

b) He then determines, from memory or a style book, the processing instructions ("controls") that will produce the format desired for that type of element.

c) Finally, he inserts the chosen controls into the text.

Here is how the start of this paper looks when marked up with controls in a typical (pre-1980) text processing formatting language:

```
.SK 1
Text processing and word processing systems typically
require additional information to be interspersed among
the natural text of the document being processed.
This added information, called "markup," serves two purposes:
.TB 4
.OF 4
.SK 1
a) Separating the logical elements of the document; and
.OF 4
.SK 1
b) Specifying the processing functions to be
performed on those elements.
.OF 0
.SK 1
```

The .SK,.TB, and .OF controls, respectively, cause the skipping of vertical space, the setting of a tab stop, and the offset, or "hanging indent", style of formatting. (The not sign (-) in each list item represents a tab code, which would otherwise not be visible.)

Procedural markup like this, however, has a number of disadvantages. For one thing, information about the document's attributes is usually lost. If the user decides, for example, to centre both headings and

[1] SGML = Standard Generalized Markup Language (ISO 8879:1986)

figure captions when formatting, the "centre" control will not indicate whether the text on which it operates is a heading or a caption. Therefore, if he wishes to use the document in an information retrieval application, search programs will be unable to distinguish headings from the text of anything else that was centred.

Procedural markup is also inflexible. If the user decides to change the style of his document (perhaps because he is using a different output device), he will need to repeat the markup process to reflect the changes. Moreover, markup with control words can be time-consuming, error-prone, and require a high degree of operator training, particularly when complex typographic results are desired. This is true (albeit less so) even when a system allows defined procedures ("macros"), since these must be added to the user's vocabulary of primitive controls.

These disadvantages of procedural markup are avoided by a markup scheme called "generalized markup" because it does not restrict documents to a single application, formatting style, or processing system. Generalized markup is based on two postulates:

a) Markup should describe a document's structure and other attributes rather than specify processing to be performed on it, as descriptive markup need be done only once and will suffice for all future processing.

b) Markup should be rigorous so that the techniques available for processing rigorously-defined objects like programs and data bases can be used for processing documents as well.

Descriptive Markup With generalized markup, the markup process stops at the first step: the user locates each significant element of the document and marks it with the mnemonic name ("generic identifier") that he feels best characterizes it. The processing system associates the markup with processing instructions in a manner that will be described shortly. SGML is the notation for generalized markup that has been standardised internationally by ISO and is being adopted throughout the world.

```
<p>
Text processing and word processing systems typically
require additional information to be interspersed among
the natural text of the document being processed
This added information, called <q>markup</q>, serves two purposes:
<ol>
<li>Separating the logical elements of the document; and
<li>Specifying the processing functions to be
performed on those elements.
</ol>
```

This example has some interesting properties:

a) There are no quotation marks in the text; the processing for the quotation element generates them and will distinguish between opening and closing quotation marks if the output device permits.

b) The comma that follows the quotation element is not actually part of it. Here, it was left outside the quotation marks during formatting, but it could just as easily have been brought inside were that style preferred.

c) There are no sequence numbers for the ordered list items; they are generated during formatting.

The source text, in other words, contains only information; characters whose only role is to enhance the presentation are generated during processing.

If, as postulated, descriptive markup like this suffices for all processing, it must follow that the processing of a document is a function of its attributes. The way text is composed offers intuitive support for this premise. Such techniques as beginning chapters on a new page, italicizing emphasized phrases, and indenting lists, are employed to assist the reader's comprehension by emphasizing the structural attributes of the document and its elements.

From this analysis, a 3-step model of document processing can be constructed:

a) Recognition: An attribute of the document is recognized, e.g., an element with a generic identifier of "footnote".

b) Mapping: The attribute is associated with a processing function. The footnote "tag", for example, could be associated with a procedure that prints footnotes at the bottom of the page or one that collects them at the end of the chapter.

c) Processing: The chosen processing function is executed.

Descriptive (or "generic") coding is a considerable improvement over procedural markup in practical use, but it is conceptually insufficient. Documents are complex objects, and they have other attributes that a markup language must be capable of describing. For example, suppose the user decides that his document is to include elements of a type called "figure" and that it must be possible to refer to individual figures by name. The markup for a particular figure element known as "angelfig" could begin with this start-tag:

```
<fig id=angelfig>
```

"Fig", of course, stands for "figure", the value of the generic identifier attribute. The GI identifies the element as a member of a set of elements having the same role. In contrast, the "unique identifier" (ID) attribute distinguishes the element from all others, even those with the same GI.

The GI and ID attributes are termed "primary" because every element can have them. There are also "secondary" attributes that are possessed only by certain element types. For example, if the user wanted some of the figures in his document to contain illustrations to be produced by an artist and added to the processed output, he could define an element type of "artwork". Because the size of the externally-generated artwork would be important, he might define artwork elements to have a secondary attribute, "depth". This would result in the following start-tag for a piece of artwork 24 picas deep:

```
<artwork depth=24p>
```

The markup for a figure would also have to describe its content. "Content" is, of course, a primary attribute, the one that the secondary attributes of an element describe. The content consists of an arrangement of other elements, each of which in turn may have other elements in its content, and so on until further division is impossible. One way in which SGML differs from generic coding schemes is in the conceptual and notational tools it provides for dealing with this hierarchical structure. These are based on the second generalized markup hypothesis, that markup can be rigorous.

Rigorous Markup Assume that the content of the figure "angelfig" consists of two elements, a figure body and a figure caption. The figure body in turn contains an artwork element, while the content of the caption is text characters with no explicit markup. The markup for this figure could look like this:

```
<fig id=angelfig>
<figbody>
<artwork depth=24p>
</artwork>
</figbody>
<figcap>Three Angels Dancing
</figcap>
</fig>
```

The markup rigorously expresses the hierarchy by identifying the beginning and end of each element in classical left list order. No additional information is needed to interpret the structure, and it would be possible to implement support by the simple scheme of macro invocation discussed earlier. The price of this simplicity, though, is that an end-tag must be present for every element.

With SGML, however, it is possible to omit much markup by advising the system about the structure and attributes of any type of element the user defines. This is done by creating a "document type definition", using a construct of the language called an "element declaration". While the markup in a document consists of descriptions of individual elements, a document type definition defines the set of all possible valid markup of a type of element.

An element declaration includes a description of the allowable content, normally expressed in a variant of regular expression notation. Suppose, for example, the user extends his definition of "figure" to permit the

figure body to contain either artwork or certain kinds of textual elements. The element declaration might look like this:

```
<!--       ELEMENTS     MIN     CONTENT (EXCEPTIONS) -->
<!ELEMENT fig           - -     (figbody, figcap?)>
<!ELEMENT figbody       - O     (artwork | (p | ol | ul)+)>
<!ELEMENT artwork       - O     EMPTY>
<!ELEMENT figcap        - O     (#PCDATA)>
```

The first declaration means that a figure contains a figure body and, optionally, can contain a figure caption following the figure body. (The hyphens will be explained shortly.)

The second says the body can contain either artwork or an intermixed collection of paragraphs, ordered lists, and unordered lists. The "O" in the markup minimization field ("MIN") indicates that the body's end-tag can be omitted when it is unambiguously implied by the start of the following element. The preceding hyphen means that the start-tag *cannot* be omitted.

The declaration for artwork defines it as having an empty content, as the art will be generated externally and merged in. As there is no content in the document, there is no need for ending markup.

The final declaration defines a figure caption's content as 0 or more characters. A character is a terminal, incapable of further division. The "O" in the "MIN" field indicates the caption's end-tag can be omitted. In addition to the reasons already given, omission is possible when the end-tag is unambiguously implied by the eng-tag of an element that contains the caption.

It is assumed that p, ol, and ul have been defined in other element declarations.

With this formal definition of figure elements available, the following markup for "angelfig" is now acceptable:

```
<fig id=angelfig>
<figbody>
<artwork depth=24p>
<figcap>Three Angels Dancing
</fig>
```

There has been a 40% reduction in markup, since the end-tags for three of the elements are no longer needed.

- As the element declaration defined the figure caption as part of the content of a figure, terminating figure automatically terminated the caption.
- Since the figure caption itself is on the same level as the figure body, the <figcap> start-tag implicitly terminated the figure body.
- The artwork element was self-terminating, as the element declaration defined its content to be empty.

A document type definition also contains an "attribute definition list declaration" for each element that has attributes. The definitions include the possible values the attribute can have, and the default value if the attribute is optional and is not specified in the document.

Here are the attribute list declarations for "figure" and "artwork".

```
<!--         ELEMENTS   NAME    VALUE   DEFAULT -->
<!ATTLIST    fig        id      ID      #IMPLIED>
<!ATTLIST    artwork    depth   CDATA   #REQUIRED>
```

The declaration for figure indicates that it can have an ID attribute whose value must be a unique identifier name. The attribute is optional and does not have a default value if not specified. In contrast, the depth attribute of the artwork element is required. Its value can be any character string. Document type definitions have uses in addition to markup minimization. They can be used to validate the markup in a document before going to the expense of processing it, or to drive prompting dialogues for users unfamiliar with a document type.

<u>Conclusion</u> Generalized markup, then, has both practical and academic benefits. In the publishing environment, it reduces the cost of markup, cuts lead times in book production, and offers maximum

flexibility from the text data base. In the office, it permits interchange between different kinds of word processors, with varying functional abilities, and allows auxiliary "documents", such as mail log entries, to be derived automatically from the relevant elements of the principal document, such as a memo.

At the same time, SGML's rigorous descriptive markup makes text more accessible for computer analysis. While procedural markup (or no markup at all) leaves a document as a character string that has no form other than that which can be deduced from analysis of the document's meaning, generalized markup reduces a document to a regular expression in a known grammar. This permits established techniques of computational linguistics and compiler design to be applied to natural language processing and other document processing applications.

PostScript: The text of this brief article is a precis of Annex A of ISO 8879:1986 which is itself derived from an article by Charles G. Goldfarb that was published in 1981.

Printed and published by the IEE, Savoy Place, London WC2R 0BL, UK.

Introduction to SGML concepts

Ellison, P.

This paper appears in: **Adding Value to Documents with Markup Languages, IEE Col**
Publication Date: 1994
On page(s): 1/1 - 1/5
Meeting Date: 06/06/1994
Location: London
INSPEC Accession Number:4765008
Posted online: 2002-08-06 19:21:06.0

Abstract
The paper provides an introduction to the concepts of SGML. Text processing systems ty
additional information to be interspersed among the natural text of the document being pı
additional information, called `markup', serves two purposes: separating the logical elem
and specifying the processing instructions to be performed on those elements. The three
procedural, generalized and descriptive, are all described. The benefits and use of SGMI
markup are also outlined

Index Terms
Inspec
Controlled Indexing
Not Available

Non-controlled Indexing
Not Available
Author Keywords
Not Available

References

No references available on IEEE Xplore.

Citing Documents

No citing documents available on IEEE Xplore.